Zappos-Challenge
Author: Ankit Singhaniya
Date: 10/03/2014

Application using REST API from Zappos
Uses: javax.json-1.0.4 library package (and requires jdk8.0.2)

To Compile in Windows, Run: javac -cp *.jar *.java
To execute, run the following: java -cp .;./javax.json-1.0.4.jar GiftSearch

In Linux, to compile, Run: javac –cp *.jar *.java
To execute, run the following: java -cp .:./javax.json-1.0.4.jar GiftSearch

Section 1: Introduction
This project proposes a solution to Zappos Challenge with the following problem statement:
"The application should take both inputs and leverage the Zappos API (http://developer.zappos.com/docs/api-documentation) to create a list of Zappos products whose combined values match as closely as possible to X dollars."

Section 1.1: Execution
The program accepts 2 command line arguments, in order:

1. Number of items in the gift
2. Total value of the gift

Note that the default values are set to $10,000.00 and 10 respectively.

Section 2: Algorithm
The algorithm combines two distinct approaches:

1. Exhaustive Search on Small Sets
2. Probabilistic Search on subset of the product range.

To choose an effective range of products for probabilistic approach, I consider the following:

1. Choose Top k products, in decreasing order of its popularity
2. Trim Products List further, as mentioned below.

Section 2.1: Trimming
The solution trims product sets according to the value of the gift and the quantity of items.

1. Rejects products of value > gift value
2. Rejects all products value < ((average gift value)/ConstantFactor)

Choice (2) trims out huge data considering with might even have made probabilistic results worse, in terms of cost of individual items in the suggestion. Naturally, we would expect the cost of items in the gift to follow some distribution as Gaussian about the average item cost.

For example,
Total Value = $10,000.0 and Number of Items = 2
Size of original Data Set (Products) with price < $10,000.00: 78199
Size of Trimmed Data Set (Products) additionally with price > $1,000: 2308

Section 2.2: Choosing Data Set
Subsets of data could be chosen in either of the two ways:

1. Choose Top k popular products
2. For the entire product prices (products ordered in decreasing order of its popularity), do the following:
   -- If product price > gift price, reject it.
   -- If already Q (quantity = number of items in gift) of (more popular) product is included, reject this product.
   -- Use trimming to further trim trailing data.

In the submission, choice (1) is made primarily because of the cost involved in making REST API calls to get all (1.2million) product details, 100 at a time. Consideration of approach (2) is one of the improvements to the data considered but involves runtime cost.

Section 2.3: Tuning
The algorithm is all about tuning the defined model parameters to gain maximum accuracy.
One way is by using normalized range of product prices. Another approach could be to increase the number of random hits.

In the submission, I have examined the model for few optimizations on the parameters and not forgetting about the Time and Memory constraints.

Section 3: Results
In cases where the set of products is small or the quantity of items in the gift is less, we go for the exhaustive search technique which guarantees to return optimal solutions. For larger inputs, the probabilistic model works best by looking only at a finite subset of data to make decisions. Since the data are repetitive in our case (product cost), it makes sense to take subset.

Below is Snapshot of the "Minimized" Output:

(A) Results for Total Value = $5,000.0 and Number of Items = 20
-- Getting Popular Products with price less than given gift price (API Calls)
-- Getting product prices finished in 00 Hrs 00 Mins 13 Secs
-- Processing: 20000 products to form gifts recommendations
-- Trimmed Products to : 14039 size
-- Processing 19449 solutions complete
-- Gift Set Generation took 00 Hrs 00 Mins 08 Secs
GIFT SET No. 1, as close as $2.15 from given price
GIFT SET No. 2, as close as $3.01 from given price
GIFT SET No. 3, as close as $3.09 from given price
GIFT SET No. 4, as close as $3.88 from given price
GIFT SET No. 5, as close as $4.05 from given price
GIFT SET No. 6, as close as $4.07 from given price
GIFT SET No. 7, as close as $5.02 from given price
GIFT SET No. 8, as close as $6.07 from given price
GIFT SET No. 9, as close as $7.17 from given price
GIFT SET No. 10, as close as $11.05 from given price
Total Execution Time: 00 Hrs 00 Mins 22 Secs

(B) Results for Total Value = $15,000.0 AND Number of Items = 90

-- Getting Popular Products with price less than given gift price (API Calls)
-- Getting product prices finished in 00 Hrs 00 Mins 10 Secs
-- Processing: 20000 products to form gifts recommendations
-- Trimmed Products to : 16753 size
-- Processing 18164 solutions complete
-- Gift Set Generation took 00 Hrs 00 Mins 10 Secs
GIFT SET No. 1, as close as $0.08 from given price
GIFT SET No. 2, as close as $0.14 from given price
GIFT SET No. 3, as close as $0.88 from given price
GIFT SET No. 4, as close as $0.93 from given price
GIFT SET No. 5, as close as $2.23 from given price
GIFT SET No. 6, as close as $3.48 from given price
GIFT SET No. 7, as close as $3.49 from given price
GIFT SET No. 8, as close as $4.39 from given price
GIFT SET No. 9, as close as $4.40 from given price
GIFT SET No. 10, as close as $4.81 from given price
Total Execution Time: 00 Hrs 00 Mins 21 Secs

Section 4: Further Improvements

Definitely, there is scope of improvement to this algorithm

1. One of it could be to attempt building a pre-learned model that caches important statistics about the data beneath it.
2. Fine tune the parameters
3. Use of Hash Maps and Heaps for efficient implementation for product lookups (as needed in approach (2) described in Section 2.2) and maintaining Result Set (in present implementation as well)