## Web Information Retrieval – Ex 3

For productSearch method we implemented the following algorithm:

```
1. Find the top C reviews that match the query using VectorSpaceSearch
   ('C' is an integer greater than 1, in our code c=30).
2. Assign each review with a weight corresponding it's ranking, where
   sum(weights)=1, and the higher the ranking – the bigger the weight.
3. Extract Product IDs of each of the above reviews. If productId is
   mentioned in more than one review, sum their weights.
4. For each productId get the posting list and find ALL the reviews
   that it appears in.
5. For each review of each productId, get the helpfulness and score. If
   helpfulness is not in range [0,1] then discard it. Then save
   (helpfulness*score) as the 'new_review_score'.
6. Calculate the average of: average('all_new_review_scores_of_pid')
   and median('all_new_review_scores_of_pid') for each product.
7. Normalize the above result for each product by the sum of all
   results.
8. Combine the weights calculated in steps 3 and 7 and normalize.
9. Return top k.
```

**Explanation of our intuition**

When trying to find the best product ID's given a query, we take 2 factors into consideration:

1. The match between a review and a query.
2. The Quality of the review and of the product discussed in the review (helpfulness and score respectively).

Our algorithm finds the top 'C' reviews that match the given query (we didn't want to find ALL reviews that partly match the query because some of them might be irrelevant as the number of matches grows. Moreover, we decided not to take a number of reviews that depends on k since it results in an inconsistent ranking of the products for different k values).

Now, given the best reviews and their rankings we want to change our point of view from reviews numbers to the product ID's discussed in those reviews, and evaluate their quality.

Next, Let 'new_score' be a value assigned for each review (of each product) based on it's reviews scores and helpfulness (We considered both factor in order to give a bigger weight to a product that has reviews that people find helpful, as well as high score. So if a review doesn't have a high helpfulness ratio it should have a lower impact on the weight).

Finally, we assigned a new weight for each product, that depends on the median and average values of the list of 'new_score' (of the reviews discussing this product).

We took both median and average to smooth potential anomalies.

Using this pair of weights (the query match weight and the score weight), we can rank the products in a relatively accurate way.

One way to improve the accuracy would be to alter step 6:
When calculating the average of ‘new_score’ of each product’s reviews, give a bigger weight for the reviews that also matched the query in step 1.