**Products Recommendation (Which two product bought together)**

In grocery shopping it frequently happens that some products are bought together than other more frequently. For example, milk and bread, bananas and strawberries.

Our idea based on this fact is to create a products recommendation list from which customer can select together with a product. Once a customer adds one product to cart, we will offer a list of recommended products to be bought together. The customer might then choose some products from the recommendation list.

If we can predict the right product bundles and offer relevant recommendation, we can help customers find the right products meanwhile boost sales and profits by providing some discounts.

To solve this problem, we need these two steps:

1. Find out which products are frequently bought together
2. Given the previous product, generate a recommendation list, and predict the next product to be bought.

**Step 1:**

For the first step, we will use [bigram](https://en.wikipedia.org/wiki/Bigram) and count bigram frequency.

A bigram is a sequence of two adjacent elements from a string of tokens.

In our case, a bigram will be the names of two products that are bought one after another.

For example, if a customer adds 'apple', 'banana', 'strawberry' to cart one by one, the bigrams will be

'apple banana', 'banana strawberry'.

Another customer adds 'banana', 'strawberry', 'milk', then the bigrams will be

'banana strawberry', 'strawberry milk'.

After getting all bigrams, we will count how many times each bigram appear.

The results of bigram frequency will be stored into a nested dictionary.

The first layer key is the first product name, and first layer value is another dictionary. The second layer keys are all second product names, and second layer values are the frequency of each first-product-second-product bigram.

If we still use the example above, the final dictionary will be:

{'apple': {'banana': 1}, 'banana': {'strawberry': 2}, {'strawberry': {'milk': 1}}

**Step 2:**

For the second step,

we will give recommendation based on the bigram frequency.

To do this, we first sort the frequencies for each bigram in decreasing order.

For example, if we have {'apple': {'strawberry': 5, 'avocado':5, 'banana': 7, 'milk': 1}}, it will be sorted as

'apple'+'banana': 7

'apple'+'avocado': 5

'apple'+'strawberry': 5

'apple'+'milk': 1

Next, we specify how many products we want to recommend. We use the parameter “k” to denote the numbers.

Then, we pick which products are in recommendation list by the following steps:

We start from the ones with highest frequency.

But there are several situations we might need to take care of.

1. What if there are several bigrams with the same frequency?

The answer to this depend on the number of recommendation k . For example, if we still use the case above, now we have 'apple'+'avocado' and 'apple'+'strawberry' the same frequency. If k= 3, we will just pick both. If k = 2 , then we will randomly pick one out of these two bundles and add to list.

1. If k is to large and there are not enough combinations to recommend?

For example, if k = 10, but in this case we only have 4 products that are ever bought together with 'apple'. We will first add all four products to recommendation list. But we still have 6 positions left. So we will start with one step further recommendation of one with highest frequency, that is to say, we will first see what can be bought together with 'banana'. If still more positions left, we will move to 'avocado', then 'strawberry', then 'milk'. If after all iterations, there are still more positions left, we will let it be that way.

**Evaluation:**

We evaluate the model on the test data by seeing how many products bought together are from the recommendation list.

For example, order 1 in test data contains 10 products.

We start by recommending what can be bought with the first product the customer added in his cart, and we will give 10 recommendations (which is of same size as the actual order). We compare the next 9 actually bought products with this 10 recommendations. If there's a match, we will add 1 to the total score (flag variable). Then we move to the second actually bought product, and give another 10 recommendations bought with the second product. Compare again, and compute total scores.

After all the iterations through all actually bought products in a order, we evaluate by dividing the total score by the order size to get the final score for a particular order number.

In the end, we calculate the average score by taking the average of all final scores for each order.

For example: if average score is 0.345, it means 34% of the products ordered by the customer in a given order are rom the recommendation list.