“Implementation of SON Algorithm”

A Report

Submitted as special assignment

of

2CS702 Big Data Analytics

By

Dhruvil Shah – 19BCE248

Rajkumar Solanki – 19BCE267

Jurin Vachhani -19BCE286

Under the Guidance of

Prof. Jigna Patel

COMPUTER SCIENCE AND ENGINEERING DEPARMENT

INSTITUTE OF TECHNOLOGY

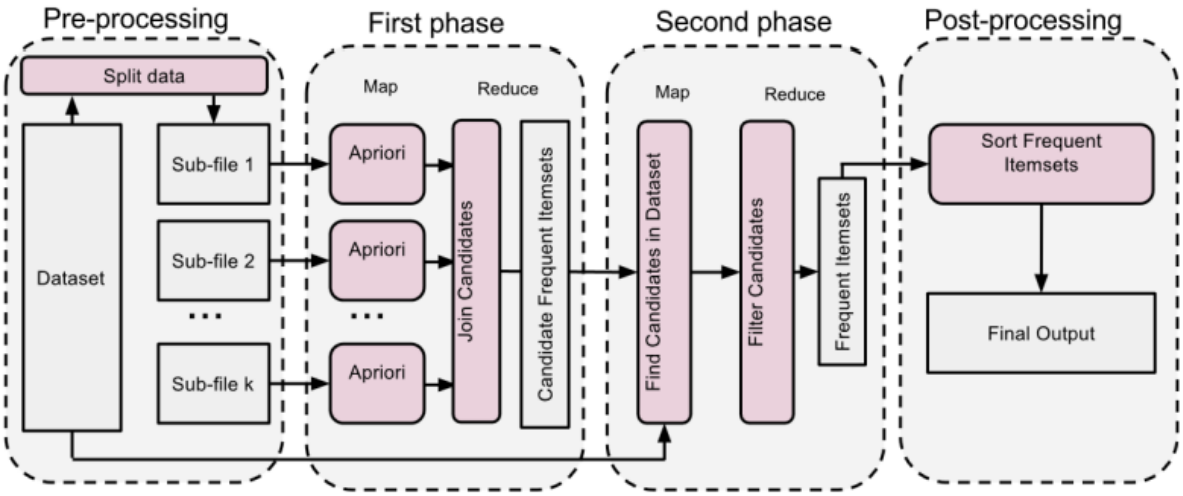
NIRMA UNIVERSITY

Ahmedabad 382 481

NOVEMBER 2022

What is SON Algorithm

- SON stands for Savasere, Omiecinski and Navathe
- SON algorithm is also known as the “Partition Algorithm”
- It basically makes a partition of the database and does testing on each of them later on combining extracted results in this way it follows MapReduce paradigm.
- It imparts itself well to a parallel-computing environment.
- Each of the chunk can be treated in parallel, and the frequent Itemsets from each chunk unite to form the candidates.
- Parallel computing and mapreduce makes the SON good for finding frequent itemsets in Big Data



About the task

- In the task we have explore the Ta Feng dataset to find the frequency itemsets.
- It is used to find the product IDs associated with a given customer ID each day.
- Aggregate all purchases a customer makes within a day into one basket. In other words, assume a customer purchases at once all items purchased within a day.

- We will create a data pipeline where the input is the raw Ta Feng data, and the output is the file described under “output file”
- We will pre-process the data, and then from this pre-processed data, we will create the final output our code is allowed to output this pre-processed data during execution, but we should not submit homework that includes this pre-processed data.
- Data preprocessing we need to generate a dataset from the Ta Feng dataset with following steps: 1. Find the date of the purchase (column TRANSACTION_DT), such as December 1, 2000 (12/1/00)
- At each date, select “CUSTOMER_ID” and “PRODUCT_ID”.
- We want to consider all items bought by a consumer each day as a separate transaction (i.e., “baskets”). For example, if consumer 1, 2, and 3 each bought oranges December 2, 2000, and consumer 2 also bought celery on December 3, 2000, we would consider that to be 4 separate transactions. An easy way to do this is to rename each CUSTOMER_ID as “DATE-CUSTOMER_ID”. For example, if COSTOMER_ID is 12321, and this customer bought apples November 14, 2000, then their new ID is “11/14/00-12321”
- Make sure each line in the CSV file is “DATE-CUSTOMER_ID1, PRODUCT_ID1”.
- The header of CSV file should be “DATE-CUSTOMER_ID, PRODUCT_ID”
- We will test our implementation with the large dataset we just generated. For this purpose, we need to report the total execution time.
- For this execution time, we take into account also the time from reading the file till writing the results to the output file.
- The following are the steps we need to do:
- 1. Reading the customer_product CSV file in to RDD and then build the case 1 market-basket model;
- 2. Find out qualified customers-date who purchased more than k items. (k is the filter threshold);

```
def renew_Candidate(preCandidate_list):
    print("preCandidate",preCandidate_list)
    #preCandidate ['101', '102', '97', '98', '99']
    res = []
    if type(preCandidate_list[0]) == str:
        for pair in combinations(preCandidate_list, 2):
            #('100', '101')
            print("Pair",pair)
            res.append(pair)
    else:
        for i in range(len(preCandidate_list)-1):
            base_tuple = preCandidate_list[i]
            for appender in preCandidate_list[i+1:]:
                if base_tuple[:-1] == appender[:-1]:
                    new_tuple = tuple(sorted(list(set(base_tuple).union(set(appender)))))
                    res.append(new_tuple)
                else:
                    break
    print("res",res)
    #res [('100', '101'), ('100', '102'), ('100', '103'), ('100', '97'), ('100', '98'),
    return res
```

- 3. Apply the SON Algorithm code to the filtered market-basket model;
- In the above slide, we can see the code for preprocessing the data.

- Here the raw_input_file is the csv file which we have taken from the kaggel and after processing it we create a new csv file data_customer_producer.csv
- It contains the information of DATE-CUSTOMER_ID and PRODUCT_ID.

```
# preprocessing
raw_input_file_path = 'ta_feng_all_months_merged.csv'
process_file_path = 'data_customer_producer.csv'
sc = SparkContext.getOrCreate()
text_rdd = sc.textFile(raw_input_file_path)
first_line = text_rdd.first()

date_customer_producer_rdd = text_rdd.filter(lambda x : x != first_line).map(lambda x : (x.split(',')[0], x.split(',')[1], x.split(',')[5])).map(l

with open(process_file_path, 'w') as output:
    writer = csv.writer(output, quoting=csv.QUOTE_NONE)
    writer.writerow(["DATE-CUSTOMER_ID", "PRODUCT_ID"])
    for row in date_customer_producer_rdd.collect():
        writer.writerow(row)
    output.close()
```

- In renew_Candidate function we will first check the length of the preCandidate list.
- If the list is not null then we have created an array named as res then we have checked the type of first element of preCandidate list if it is string then we combine the preCandidate list and append the res.

```
def get_candidate_itemsets(subset, support, total_data_size):
    if subset is None:
        return
    baskets_list = list(subset)
    scaled_down_sp = support * float(len(baskets_list) / total_data_size)
    single_item_dict = {}
    final_dict = {}
    for basket in baskets_list:
        for item in basket:
            if item not in single_item_dict.keys():
                single_item_dict[item] = 1
            else:
                single_item_dict[item] += 1
    filtered_item_dict = dict(filter(lambda item_count : item_count[1] >= scaled_down_sp, single_item_dict.items()))
    frequent_single_item_list = list(filtered_item_dict.keys())
    candidate_list = frequent_single_item_list
    curLen = 1
    while candidate_list is not None and len(candidate_list)>0:
        item_count_dict = {}
        for basket in baskets_list:
            basket = list(set(basket).intersection(set(frequent_single_item_list)))
            for candidate in candidate_list:
                curSet = set()
                if type(candidate) == str:
                    curSet.add(candidate)
                else:
                    curSet = set(candidate)
                if curSet.issubset(set(basket)):
                    if candidate not in item_count_dict.keys():
                        item_count_dict[candidate] = 1
                    else:
                        item_count_dict[candidate] += 1
        filtered_item_dict = dict(filter(lambda item_count: item_count[1] >= scaled_down_sp, item_count_dict.items()))
        final_dict[curLen] = list(filtered_item_dict.keys())
        curLen += 1
        candidate_list = renew_Candidate(sorted(list(filtered_item_dict.keys())))
    return final_dict.values()
```

- In get_candidate_itemsets where the subsets are checked if it is null then it will return.
- In the basket_list we assign list of subsets then the scaled_down_sp is calculated with a formula in the code which is actually a threshold value.
- Now the item_count_dict, filter_item_count and final_dict_values is been calculated.

```python
def count_itemsets(subset,candidates):
    item_count_dict = {}
    baskets_set = set()
    basket = list(subset)
    for ele in basket:
        baskets_set.add(ele)
    for candidate in candidates:
        curSet = set()
        if type(candidate) == str:
            curSet.add(candidate)
        else:
            curSet = set(candidate)
        if curSet.issubset(baskets_set):
            if candidate not in item_count_dict.keys():
                item_count_dict[candidate] = 1
            else:
                item_count_dict[candidate] += 1
    return item_count_dict.items()
```

```python
def normalize(lists):
    res_dict = collections.defaultdict(list)
    for item in lists:
        if type(item) == str:
            item = tuple(item.split(','))
            item = str(str(item)[:-2] + ')')
            res_dict[1].append(item)
        else:
            item = sorted(item)
            value = str(tuple(item))
            res_dict[len(item)].append(value)

    for k,v in res_dict.items():
        res_dict[k] = sorted(v)
    return res_dict
```

```
start_time = time.time()
filter_threshold = 20
support_num = 50
input_file_path = "date_customer_producer.csv"
output_file_path = "output2.txt"
text_rdd = sc.textFile(input_file_path,6)
first_line = text_rdd.first()
baskets_rdd = None
```

- Now we have set the filter_threshold value 20 and support_num value 50.
- Taking the input from the data_customer_producer.csv file after preprocessing and creating an output file output2.txt
- Lastly by printing the duration of the whole execution process.

Conclusion:

By getting the frequent itemset we can see use in real life application like e-commerce website likes amazon, flipkart, myntra where user gets recommended products based on the items selected currently. So by deploying these code onto the server this services can be made use of.