CSC 440 Data Mining Project Report (roadmap)

Student: Yangxin Fan
Date: 10/14/2020

Part I: Data Preprocessing

Step one: Read the UCI Adult Census Dataset: we have 48842 tuples × 15 attributes. In terms of attributes, we have 9 categorical attributes.

We can think of each tuple as a "transaction". Each transaction can have different values for its attributes. We want to find all frequent patterns or item sets in the UCI database.

Step two: For simplicity and interpretability, we only analyze these 9 categorical attributes described by works. Since for those numerical attributes, for example age and hours-per-week can be in same value range, we cannot differentiate them easily. Now, we get a numpy array of categorical attributes and their values. Set the min_sup = 20000

Step three: Find all items in transactions.

Part II: Apriori [AS94b]

Find frequent itemsets using an iterative level-wise approach based on candidate generation

Step one: Find frequent 1-itemset.

Step two: Build functions has_infrequent and apriori_gen:

1. Function has_infrequent: Find whether a candidate itemset has an infrequent subset, if it has, then I would not be added (superset of an infrequent itemset must be infrequent)
2. Function apriori_gen: generate candidates of frequent item sets

Step three: Call above two functions iteratively to generate candidates of frequent item sets and check whether their counts are above min_sup or not. And then add those whose counts greater than min_sup.

Part III: FP-growth [HPY00]

Mine frequent itemsets using an FP-tree by pattern fragment growth

Step one: First database scan: Acquire a list of frequent items and their support counts and sort it by counts

Step two: Construct FP-Tree (Second database scan):

1. Create FP-Tree node data structure
2. Create a sorted item list according to counts and a similarity item dictionary that could link different nodes with same item name together

3. Create function similar_item_table_update, fp_tree_create_and_update to link nodes and update counts of nodes and use class fpTreeNode to create new nodes Eventually, to build a FP-Tree

Step three: Mine FP-Tree and generate frequent patterns:

1. Create function create_cond_base to create conditional pattern base
2. Create function_fptree to create condition trees from conditional bases

Part IV: Improved version of Apriori Algorithm

I accomplished three improvements compared to original Apriori Algorithm:

1. Less scan of transactions in database: if a transaction does not contain any frequent k itemset, I will remove such a transaction. Hence, it is not reconsidered when finding the counts of k+1 candidate frequent itemset. Punchline is that: if a transaction does not contain any frequent k itemset, it cannot have frequent k+1 itemset.

2. When pruning candidate itemset, we do not need to check the two itemset where the specific candidate generated from since these two itemset are already frequent.

3. Delete transactions where their length less than the length of candidate itemset we want to check for frequency