Comparing Apriori Algorithm with a Brute Force Approach for Association Mining

Abhijeet Sahdev (as4673)
Submitted to: Dr. Jason T.L. Wang

October 12, 2024

Contents

1 Oracle SQl Statements

Create table statements implemented using Oracle SQL Developer and VSC.

```
DROP TABLE Transactions1;
DROP TABLE Transactions2;
DROP TABLE Transactions3;
DROP TABLE Transactions4;
DROP TABLE Transactions5;

CREATE TABLE Transactions1 (
    ID NUMBER GENERATED ALWAYS AS IDENTITY PRIMARY KEY,
    Items VARCHAR2(4000)
);

CREATE TABLE Transactions2 (
    ID NUMBER GENERATED ALWAYS AS IDENTITY PRIMARY KEY,
    Items VARCHAR2(4000)
);

CREATE TABLE Transactions3 (
    ID NUMBER GENERATED ALWAYS AS IDENTITY PRIMARY KEY,
    Items VARCHAR2(4000)
);

CREATE TABLE Transactions4 (
    ID NUMBER GENERATED ALWAYS AS IDENTITY PRIMARY KEY,
    Items VARCHAR2(4000)
);

CREATE TABLE Transactions5 (
    ID NUMBER GENERATED ALWAYS AS IDENTITY PRIMARY KEY,
    Items VARCHAR2(4000)
);

INSERT INTO Transactions1 (Items) VALUES
   ('Deodorants,Tissues,Wipes');
INSERT INTO Transactions1 (Items) VALUES ('Shower Gel,Extension
   cable');
INSERT INTO Transactions1 (Items) VALUES ('Face Wash,Security
   camera,Humidifier');
INSERT INTO Transactions1 (Items) VALUES
   ('Adaptors,Mouse,Supplements');
INSERT INTO Transactions1 (Items) VALUES ('Water Jug,Filters,
   Humidifier');
```

```
INSERT INTO Transactions1 (Items) VALUES ('Privacy Screen
   Protector,Tape,Pillows');
INSERT INTO Transactions1 (Items) VALUES ('Laptop case,Deodorant
   Stick');
INSERT INTO Transactions1 (Items) VALUES ('Deodorants,Extension
   cable,Security amera');
INSERT INTO Transactions1 (Items) VALUES ('Tissues,Wipes,Shower
   Gel');
INSERT INTO Transactions1 (Items) VALUES ('Face
   Wash,Mouse,Adaptors');
INSERT INTO Transactions1 (Items) VALUES ('Humidifier,Water
   Jug,Tape');
INSERT INTO Transactions1 (Items) VALUES
   ('Supplements,Filters,Pillows');
INSERT INTO Transactions1 (Items) VALUES ('Privacy Screen
   Protector,Laptop case');
INSERT INTO Transactions1 (Items) VALUES ('Deodorant
   Stick,Deodorants');
INSERT INTO Transactions1 (Items) VALUES ('Tissues,Shower
   Gel,Extension cable');
INSERT INTO Transactions1 (Items) VALUES ('Wipes,Security camera');
INSERT INTO Transactions1 (Items) VALUES ('Face
   Wash,Humidifier,Mouse');
INSERT INTO Transactions1 (Items) VALUES
   ('Adaptors,Supplements,Water Jug');
INSERT INTO Transactions1 (Items) VALUES ('Filters,Tape,Privacy
   Screen Protector');
INSERT INTO Transactions1 (Items) VALUES ('Pillows,Laptop
   case,Deodorant Stick');
INSERT INTO Transactions1 (Items) VALUES
   ('Deodorants,Tissues,Security camera');
INSERT INTO Transactions1 (Items) VALUES ('Wipes,Shower Gel,Mouse');
INSERT INTO Transactions1 (Items) VALUES ('Extension cable,Face
   Wash,Adaptors');
INSERT INTO Transactions1 (Items) VALUES
   ('Humidifier,Supplements,Filters');
INSERT INTO Transactions1 (Items) VALUES ('Water Jug,Privacy Screen
   Protector,Tape');
INSERT INTO Transactions1 (Items) VALUES ('Pillows,Deodorant
   Stick,Laptop case');
INSERT INTO Transactions1 (Items) VALUES ('Deodorants,Shower Gel,
   Humidifier');
INSERT INTO Transactions1 (Items) VALUES ('Tissues,Wipes,Mouse');
INSERT INTO Transactions1 (Items) VALUES ('Security
   camera,Humidifier,Adaptors');
INSERT INTO Transactions1 (Items) VALUES ('Face Wash,Supplements,
   Humidifers');
```

```
INSERT INTO Transactions2 (Items) VALUES ('Deodorants, Security
   camera, Tissues');
INSERT INTO Transactions2 (Items) VALUES ('Tissues');
INSERT INTO Transactions2 (Items) VALUES ('Wipes, Shower Gel,
   Extension cable');
INSERT INTO Transactions2 (Items) VALUES ('Shower gel, Extension
   cable');
INSERT INTO Transactions2 (Items) VALUES ('Extension cable');
INSERT INTO Transactions2 (Items) VALUES ('Face wash');
INSERT INTO Transactions2 (Items) VALUES ('Security camera');
INSERT INTO Transactions2 (Items) VALUES ('Deodorants, Tissues');
INSERT INTO Transactions2 (Items) VALUES ('Deodorants, Wipes');
INSERT INTO Transactions2 (Items) VALUES ('Deodorants, Shower gel');
INSERT INTO Transactions2 (Items) VALUES ('Tissues, Wipes');
INSERT INTO Transactions2 (Items) VALUES ('Tissues, Shower gel');
INSERT INTO Transactions2 (Items) VALUES ('Wipes, Shower gel');
INSERT INTO Transactions2 (Items) VALUES ('Extension cable, Face
   wash');
INSERT INTO Transactions2 (Items) VALUES ('Extension cable, Security
   Camera');
INSERT INTO Transactions2 (Items) VALUES ('Face wash, Security
   Camera');
INSERT INTO Transactions2 (Items) VALUES ('Deodorants, Extension
   cable');
INSERT INTO Transactions2 (Items) VALUES ('Tissues, Face wash');
INSERT INTO Transactions2 (Items) VALUES ('Wipes, Security Camera');
INSERT INTO Transactions2 (Items) VALUES ('Shower gel, Extension
   cable');

INSERT INTO Transactions3 (Items) VALUES ('Deodorants, Face wash');
INSERT INTO Transactions3 (Items) VALUES ('Tissues, Security
   Camera');
INSERT INTO Transactions3 (Items) VALUES ('Wipes, Extension cable');
INSERT INTO Transactions3 (Items) VALUES ('Shower gel, Face wash');
INSERT INTO Transactions3 (Items) VALUES ('Deodorants, Security
   Camera');
INSERT INTO Transactions3 (Items) VALUES ('Tissues, Extension
   cable');
INSERT INTO Transactions3 (Items) VALUES ('Wipes, Face wash');
INSERT INTO Transactions3 (Items) VALUES ('Shower gel, Security
   Camera');
INSERT INTO Transactions3 (Items) VALUES ('Deodorants');
INSERT INTO Transactions3 (Items) VALUES ('Tissues');
INSERT INTO Transactions3 (Items) VALUES ('Wipes');
INSERT INTO Transactions3 (Items) VALUES ('Shower gel');
INSERT INTO Transactions3 (Items) VALUES ('Extension cable');
```

```
INSERT INTO Transactions3 (Items) VALUES ('Face wash');
INSERT INTO Transactions3 (Items) VALUES ('Security camera');
INSERT INTO Transactions3 (Items) VALUES ('Deodorants, Tissues');
INSERT INTO Transactions3 (Items) VALUES ('Deodorants, Wipes');
INSERT INTO Transactions3 (Items) VALUES ('Tissues, Shower gel');
INSERT INTO Transactions3 (Items) VALUES ('Wipes, Extension cable');
INSERT INTO Transactions3 (Items) VALUES ('Shower gel, Face wash');

INSERT INTO Transactions4 (Items) VALUES ('Deodorants, Face wash');
INSERT INTO Transactions4 (Items) VALUES ('Tissues, Security
   Camera');
INSERT INTO Transactions4 (Items) VALUES ('Wipes, Extension cable');
INSERT INTO Transactions4 (Items) VALUES ('Shower gel, Face wash');
INSERT INTO Transactions4 (Items) VALUES ('Deodorants, Security
   Camera');
INSERT INTO Transactions4 (Items) VALUES ('Tissues, Extension
   cable');
INSERT INTO Transactions4 (Items) VALUES ('Wipes, Face wash');
INSERT INTO Transactions4 (Items) VALUES ('Shower gel, Security
   Camera');
INSERT INTO Transactions4 (Items) VALUES ('Deodorants');
INSERT INTO Transactions4 (Items) VALUES ('Tissues');
INSERT INTO Transactions4 (Items) VALUES ('Wipes');
INSERT INTO Transactions4 (Items) VALUES ('Shower gel');
INSERT INTO Transactions4 (Items) VALUES ('Extension cable');
INSERT INTO Transactions4 (Items) VALUES ('Face wash');
INSERT INTO Transactions4 (Items) VALUES ('Security camera');
INSERT INTO Transactions4 (Items) VALUES ('Deodorants, Tissues');
INSERT INTO Transactions4 (Items) VALUES ('Deodorants, wipes');
INSERT INTO Transactions4 (Items) VALUES ('Tissues, Shower gel');
INSERT INTO Transactions4 (Items) VALUES ('Wipes, Extension cable');
INSERT INTO Transactions4 (Items) VALUES ('Shower gel, Face wash');

INSERT INTO Transactions5 (Items) VALUES ('Wipes, Security Camera');
INSERT INTO Transactions5 (Items) VALUES ('Shower gel, Extension
   cable');
INSERT INTO Transactions5 (Items) VALUES ('Deodorants, Face wash');
INSERT INTO Transactions5 (Items) VALUES ('Tissues, Security
   Camera');
INSERT INTO Transactions5 (Items) VALUES ('Wipes, Extension cable');
INSERT INTO Transactions5 (Items) VALUES ('Shower gel, Face wash');
INSERT INTO Transactions5 (Items) VALUES ('Deodorants, Security
   Camera');
INSERT INTO Transactions5 (Items) VALUES ('Tissues, Extension
   cable');
INSERT INTO Transactions5 (Items) VALUES ('Wipes, Face wash');
```

```
INSERT INTO Transactions5 (Items) VALUES ('Shower gel, Security
   Camera');
INSERT INTO Transactions5 (Items) VALUES ('Deodorants');
INSERT INTO Transactions5 (Items) VALUES ('Tissues');
INSERT INTO Transactions5 (Items) VALUES ('Wipes');
INSERT INTO Transactions5 (Items) VALUES ('Shower gel');
INSERT INTO Transactions5 (Items) VALUES ('Extension cable');
INSERT INTO Transactions5 (Items) VALUES ('Face wash');
INSERT INTO Transactions5 (Items) VALUES ('Security camera');
INSERT INTO Transactions5 (Items) VALUES ('Deodorants, Tissues');
INSERT INTO Transactions5 (Items) VALUES ('Deodorants, wipes');
INSERT INTO Transactions5 (Items) VALUES ('Tissues, Shower gel');
INSERT INTO Transactions5 (Items) VALUES ('Wipes, Extension cable');
```

2 Implementation

The codes for implementing Apriori and Brute Force ($F_(K-1)XF_(K)$) algorithms.

```
import oracledb
import time
from itertools import combinations
from collections import Counter
import tracemalloc

class AssociationMining:
    """
    A class for performing association rule mining using Apriori and
       brute-force algorithms.
    """
    def __init__(self, minsup, minconf):
        """
        Initializes AssociationMining with minimum support and
           confidence.

        Args:
            minsup (float): The minimum support threshold.
            minconf (float): The minimum confidence threshold.
        """
        self.minsup = minsup
        self.minconf = minconf
        self.transactions = []

    @staticmethod
    def load_data_from_db():
        """
        Loads transaction data from an Oracle database.

        Returns:
            dict: A dictionary where keys are table names and values
               are lists of transactions.
        """
        username = "as4673"  # Replace with your Oracle username
        password = "****"  # Replace with your Oracle password
        connection_string = "(DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)
        (HOST=prophet.njit.edu)(PORT=1521))
        (CONNECT_DATA=(SID=course)))"  # Replace with your
           connection string

        results_dict = {}
        tables = ['Transactions1', 'Transactions2', 'Transactions3',
           'Transactions4', 'Transactions5']
```

```
        try:
            connection = oracledb.connect(user=username,
               password=password, dsn=connection_string)  #
               Establish database connection
            cursor = connection.cursor()  # Create a cursor object

            for table in tables:
                query = f"SELECT * FROM {table}"  # SQL query to
                   fetch all data from the table
                cursor.execute(query)  # Execute the query
                results = cursor.fetchall()  # Fetch all results
                results_dict[table] = [row[1].split(',') for row in
                   results]  # Extract transaction data (assuming
                   comma-separated items)

            return results_dict

        except oracledb.Error as error:
            print(f"Error connecting to database: {error}")  # Print
               error message if connection fails
            return {}

        finally:
            if 'cursor' in locals():
                cursor.close()  # Close the cursor if it exists
            if 'connection' in locals():
                connection.close()  # Close the database connection
                   if it exists

    def measure_performance(self, transactions, algorithm='apriori'):
        """
        Measure the performance of the specified algorithm in terms
           of time and memory.

        Args:
            transactions (list): A list of transactions.
            algorithm (str, optional): The algorithm to use
               ('apriori' or 'bruteforce'). Defaults to 'apriori'.

        Returns:
            list: The association rules generated by the algorithm.
        """
        start_time = time.time()  # Record the start time
        tracemalloc.start()  # Start memory usage tracking

        if algorithm == 'apriori':
```

```
            rules = self.apriori(transactions)  # Run the Apriori
                algorithm
        elif algorithm == 'bruteforce':
            rules = self.bruteforce(transactions)  # Run the
                brute-force algorithm
        else:
            raise ValueError("Invalid algorithm name. Choose
                'apriori' or 'bruteforce'.")  # Raise error for
                invalid algorithm

        current, peak = tracemalloc.get_traced_memory()  # Get
            current and peak memory usage
        tracemalloc.stop()  # Stop memory usage tracking
        end_time = time.time()  # Record the end time

        # Print performance metrics
        print(f"\n{algorithm.capitalize()} Performance:")
        print(f"Time taken: {end_time - start_time:.5f} seconds")
        print(f"Peak memory usage: {peak / 10**6:.5f} MB")

        return rules

    def apriori(self, transactions):
        """
        Generates association rules using the Apriori algorithm.

        Args:
            transactions (list): A list of transactions.

        Returns:
            list: The association rules generated by the Apriori
                algorithm.
        """
        num_transactions = len(transactions)
        # Count item occurrences
        item_counts = Counter(item for transaction in transactions
            for item in set(transaction))
        # Generate frequent 1-itemsets
        frequent_itemsets = [{frozenset([item]): count for item,
            count in item_counts.items() if count / num_transactions
            >= self.minsup}]

        k = 2
        while frequent_itemsets[-1]:  # Continue until no more
            frequent itemsets are found
            candidates =
                set(self.candidate_gen(frequent_itemsets[-1], k-1))
```

```
            # Generate candidate itemsets
            candidate_counts = Counter()  # Count candidate
                occurrences
            for transaction in transactions:
                for candidate in candidates:
                    if candidate.issubset(transaction):
                        candidate_counts[candidate] += 1
            # Prune candidates that don't meet minimum support
            frequent_itemsets.append({cand: count for cand, count in
                candidate_counts.items() if count / num_transactions
                >= self.minsup})
            k += 1

        all_rules = []
        for level in frequent_itemsets[1:]:
            for itemset in level:
                rules_from_itemset = self.ap_genrules(itemset,
                    frozenset(), level[itemset], num_transactions)
                all_rules.extend(rules_from_itemset)  # Collect all
                    rules

        return all_rules

    def ap_genrules(self, itemset, Hm, support_count,
        num_transactions):
        """
        Recursively generates association rules from frequent
            itemsets.

        Args:
            itemset (frozenset): The frequent itemset.
            Hm (frozenset): The current set of consequents.
            support_count (int): The support count of the itemset.
            num_transactions (int): The total number of transactions.
            rules (list): The list to store generated rules.
        """
        rules = []
        m = len(Hm)
        if len(itemset) > m + 1:
            Hmp1 = self.candidate_gen([Hm], m)
            Hmp1 = {h for h in Hmp1 if all(frozenset(h - {item}) in
                self.frequent_itemsets[m] for item in h)}
            for h in Hmp1:
                conf = support_count / self.get_support(itemset - h,
                    num_transactions)
                if conf >= self.minconf:
                    # Create rule directly here and add to local list
```

```
                    rules.append((tuple(itemset - h), tuple(h),
                       support_count / num_transactions, conf))

        # Recursively generate more rules using lists
        for rule in list(rules):
            if len(rule[1]) < len(itemset) - 1:
                new_Hm = frozenset(rule[1])
                more_rules = self.ap_genrules(itemset, new_Hm,
                   support_count, num_transactions)
                rules.extend(more_rules)

        return rules

    def candidate_gen(self, itemsets, k):
        """
        Generates candidate itemsets of size k+1 from frequent
           itemsets of size k.

        Args:
            itemsets (dict): A dictionary of frequent itemsets and
               their counts.
            k (int): The size of the current frequent itemsets.

        Returns:
            set: A set of candidate itemsets.
        """
        candidates = set()
        for a in itemsets:
            for b in itemsets:
                if len(a.union(b)) == k + 1:  # Check if the union
                   results in a k+1 itemset
                    candidate = a.union(b)
                    # Check if all subsets of the candidate are
                       frequent
                    if all(self.has_frequent_subset(candidate,
                       frozenset(), k, itemsets) for _ in
                       range(k+1)):
                        candidates.add(candidate)  # Add the
                           candidate if all subsets are frequent
        return candidates

    def has_frequent_subset(self, candidate, Hm, k,
       frequent_itemsets):
        """
        Checks if all k-sized subsets of a candidate itemset are
           frequent.
```

```
        Args:
            candidate (frozenset): The candidate itemset.
            Hm (frozenset): The current set of consequents (not used
               here).
            k (int): The size of the subsets to check.
            frequent_itemsets (dict): A dictionary of frequent
               itemsets and their counts.

        Returns:
            bool: True if all k-sized subsets are frequent, False
               otherwise.
        """
        if k == 1:  # Base case: all 1-itemsets are frequent
            return True
        for subset in combinations(candidate, k):  # Generate all
           k-sized subsets
            if frozenset(subset) not in frequent_itemsets:  # Check
               if the subset is frequent
                return False  # Return False if any subset is not
                   frequent
        return True  # Return True if all subsets are frequent


    def bruteforce(self, transactions):
        """
         Generates association rules using K-1 K approach
        """
        all_items = set(item for transaction in transactions for
           item in transaction)
        num_transactions = len(transactions)
        max_k = len(all_items) + 1  # max possible size of itemset

        frequent_itemsets = []

        for k in range(1, max_k):
            k_itemsets = [
                frozenset(itemset) for itemset in
                   combinations(all_items, k)
                if sum(1 for t in transactions if
                   frozenset(itemset).issubset(t)) /
                   num_transactions >= self.minsup
            ]
            if not k_itemsets:
                break
            frequent_itemsets.extend(k_itemsets)

        rules = []
```

```
        for itemset in frequent_itemsets:
            if len(itemset) > 1:
                for rhs_size in range(1, len(itemset)):
                    for rhs in combinations(itemset, rhs_size):
                        lhs = itemset - set(rhs)
                        support_count = self.get_support(itemset,
                           transactions)
                        confidence = support_count /
                           self.get_support(frozenset(lhs),
                           transactions)
                        if confidence >= self.minconf:
                            support = support_count /
                               num_transactions
                            rules.append((tuple(lhs), tuple(rhs),
                               support, confidence))
        return rules

    def get_support(self, itemset, transactions):
        itemset_set = set(itemset) # For faster operations
        count = 0
        for transaction in transactions:
            if itemset_set.issubset(set(transaction)):
                count += 1
        return count


if __name__ == "__main__":
    minsup = 0.5
    minconf = 0.5
    mining = AssociationMining(minsup, minconf)
    results_dict = mining.load_data_from_db()

    for table, transactions in results_dict.items():
        print(f"\nAnalyzing Table: {table}")
        for method in ['apriori', 'bruteforce']:
            rules = mining.measure_performance(transactions, method)
            print(f"\nNumber of {method} rules: {len(rules)}")
            for rule in rules:
                print(f"{set(rule[0])} => {set(rule[1])} (Support:
                   {rule[2]:.2f}, Confidence: {rule[3]:.2f})")
```

2.1 Screenshots

Output of the code from my terminal (iterm2) for minsup = 0.5, minconf = 0.5.