Association Rule Mining Project Report

Name: Dathwik Kollikonda
NJIT UCID: dk649
Email Address: dk649@njit.edu
Date: 03/02/2025
Professor: Yasser Abduallah
Course: CS 634854 Data Mining

Abstract

- This project implements and evaluates two association rule mining algorithms: Brute Force and Apriori. Using retail transaction data from five stores—Amazon, Best Buy, Nike, K-Mart, and Target—we analyze their effectiveness in identifying purchasing patterns and generating association rules. Through this exploration, the project highlights the practical application of data mining techniques in a retail setting, offering insights into algorithm performance, customer behaviour, and the trade-offs between different approaches to association rule mining.

Introduction

- Association rule mining is a fundamental data analytics technique, especially useful in the retail and e-commerce sectors. This project aims to implement and compare two key algorithms:
  1. **Brute Force**: A custom implementation that exhaustively checks all possible itemsets.
  2. **Apriori**: An efficient algorithm for frequent itemset generation, implemented using the mlxtend library.
- These algorithms are applied to transaction data from five major retailers—Amazon, Best Buy, Nike, K-Mart, and Target. By evaluating their performance and outcomes, this project seeks to uncover the strengths and limitations of each approach in retail data analysis.

Core Concepts and Principles

- **Frequent Itemset Discovery:**
  - At the heart of association rule mining is the discovery of item sets that frequently appear together in transactions. This project explores and compares different approaches to this task, ranging from the exhaustive brute force method to more efficient algorithms like Apriori.
- **Support and Confidence Metrics:**

Two key metrics guide our analysis:

Support: Measures how frequently an itemset appears in the dataset.

Confidence: Assesses the likelihood of an item being purchased given the purchase of another item.

These metrics are crucial for filtering and ranking association rules.

- **Association Rules:**

The main goal of this project is to identify meaningful association rules that reveal patterns in customer purchasing behaviour. These insights can be applied to various business strategies, such as enhancing product recommendations and optimizing store layouts.

- **Algorithm Efficiency:**

A crucial aspect of this project is evaluating the computational efficiency of different algorithms. By measuring and comparing execution times, we analyze the trade-offs between exhaustive search methods and more optimized approaches.

- **Data Preprocessing:**

  The project includes steps for loading and preprocessing transaction data, demonstrating the importance of data preparation in data mining tasks.

Project Workflow

1. **Data Preparation and Loading:**
   - CSV File Creation: Transaction data provided by the professor was expanded and organized into Excel sheets for each store (Amazon, Best Buy, Nike, K-mart, and Target). Target was not given as an example it was done manually.
   - CSV Loading: A custom function (read_transactions_from_csv) was implemented to read these CSV files and convert them into a format suitable for analysis.
2. **User Interaction:**
   - The program implements a user-friendly interface allowing selection of the store to analyze and input of support and confidence thresholds.
   - Input validation functions ensure that user inputs are within acceptable ranges.
3. **Algorithm Implementation:**

   a. Brute Force Algorithm:
   - Custom implementation using Python's itertools for combination generation.
   - Exhaustively checks all possible itemsets against the minimum support threshold.
   - Generates association rules based on frequent itemsets and minimum confidence.

   b. Apriori Algorithm:

   - Utilizes the mlxtend library's implementation of Apriori.
   - Efficiently generates frequent itemsets using the apriori principle.
4. **Performance Measurement:**
   - Execution time is measured for each algorithm using Python's time module.
   - Allows for direct comparison of algorithmic efficiency.
5. **Result Comparison and Analysis:**
   - Compares the number of frequent itemsets and rules generated by each algorithm.
   - Checks for consistency in results across both methods.
   - Identifies the fastest algorithm for each analysis.

6. **Rule Display and Interpretation:**
    - Implements functions to display association rules in a readable format.
    - Provides insights into the strength of associations through support and confidence metrics.
7. **Item Analysis:**
    - Calculates and displays item counts and support values.
    - Indicates whether individual items meet the specified support threshold.

Implementation Details

Data Handling

- Utilized pandas for data manipulation
- Implemented custom functions for CSV file operations

Brute Force Method

- Iteratively generated candidate itemsets
- Calculated support and confidence for each potential rule

Library Methods

- Used mlxtend's implementations of Apriori
- Leveraged pandas DataFrames for efficient data processing

Performance Measurement

- Used Python's time module to measure execution time for each algorithm

Results and Analysis

When analyzing the Nike store transactions with a minimum support of 55% and minimum confidence of 70%, we obtained the following results:

1. **Brute Force Method:**
    - Generated 6 association rules
    - Execution time: 0.0021 seconds
2. **Apriori Algorithm:**
    - Generated 6 association rules
    - Execution time: 0.0127 seconds

Sample rules generated:

Rule 1: {'Socks'} -> {'Running Shoe'} (Confidence: 0.85, Support: 0.55)
Rule 2: {'Running Shoe'} -> {'Socks'} (Confidence: 0.79, Support: 0.55)
Rule 3: {'Running Shoe'} -> {'Sweatshirts'} (Confidence: 0.79, Support: 0.55)
Rule 4: {'Sweatshirts'} -> {'Running Shoe'} (Confidence: 0.85, Support: 0.55)
Rule 5: {'Socks'} -> {'Sweatshirts'} (Confidence: 0.92, Support: 0.60)

Rule 6: {'Sweatshirts'} -> {'Socks'} (Confidence: 0.92, Support: 0.60)

Comparison of Algorithms

- Our analysis showed that while both the methods produced the same number of rules for the given parameters, their execution times varied:

    1. Brute Force Method: Was the fastest, provides a baseline for comparison
    2. Apriori Algorithm: Was the slowest

Conclusion

- This project successfully implemented and compared two different approaches to association rule mining. While the custom brute force method offers valuable insight into the underlying process, the association rules derived from our retail transaction data provide meaningful insights into customer purchasing behavior. These findings can be applied to product placement, marketing strategies, and inventory management.

Screenshots

My CSV files look like this:

	A	B	C	D	E	F	G	H	I	J
1	Desk Top	Printer	Flash Drive	Microsoft Office	Speakers	Anti-Virus				
2	Lab Top	Flash Drive	Microsoft Office	Lab Top Case	Anti-Virus					
3	Lab Top	Printer	Flash Drive	Microsoft Office	Anti-Virus	Lab Top Case	External Hard-Drive			
4	Lab Top	Printer	Flash Drive	Anti-Virus	External Hard-Drive	Lab Top Case				
5	Lab Top	Flash Drive	Lab Top Case	Anti-Virus						
6	Lab Top	Printer	Flash Drive	Microsoft Office						
7	Desk Top	Printer	Flash Drive	Microsoft Office						
8	Lab Top	External Hard-Drive	Anti-Virus							
9	Desk Top	Printer	Flash Drive	Microsoft Office	Lab Top Case	Anti-Virus	Speakers	External Hard-Drive		
10	Digital Camera	Lab Top	Desk Top	Printer	Flash Drive	Microsoft Office	Lab Top Case	Anti-Virus	External H	Speakers
11	Lab Top	Desk Top	Lab Top Case	External Hard-Drive	Speakers	Anti-Virus				
12	Digital Camera	Lab Top	Lab Top Case	External Hard-Drive	Anti-Virus	Speakers				
13	Digital Camera	Speakers								
14	Digital Camera	Desk Top	Printer	Flash Drive	Microsoft Office					
15	Printer	Flash Drive	Microsoft Office	Anti-Virus	Lab Top Case	Speakers	External Hard-Drive			
16	Digital Camera	Flash Drive	Microsoft Office	Anti-Virus	Lab Top Case	External Hard-Drive	Speakers			
17	Digital Camera	Lab Top	Lab Top Case							
18	Digital Camera	Lab Top Case	Speakers							
19	Digital Camera	Lab Top	Printer	Flash Drive	Microsoft Office	Speakers	Lab Top Case	Anti-Virus		
20	Digital Camera	Lab Top	Speakers	Anti-Virus	Lab Top Case					
21										

Screenshot 1: Best Buy Transactions

Following screenshots are snippets from my code:

```python
import pandas as pd
import itertools
from collections import defaultdict
import time
from mlxtend.frequent_patterns import apriori as apriori_lib
from mlxtend.frequent_patterns import association_rules
from mlxtend.preprocessing import TransactionEncoder

# Function to read transactions from a CSV file
def read_transactions_from_csv(file_path):
    data_frame = pd.read_csv(file_path, header=None)
    return [set(str(item) for item in transaction if pd.notna(item)) for transaction in data_frame.values.tolist()]

# Function to ensure valid float input for support and confidence
def get_valid_float(prompt, min_value, max_value):
    while True:
        try:
            user_input = float(input(prompt))
            if min_value <= user_input <= max_value:
                return user_input / 100  # Convert percentage to decimal
            else:
                print(f"Please provide a value between {min_value} and {max_value}.")
        except ValueError:
            print("Invalid input. Please enter a valid number.")

# Function to ensure valid integer input for store selection
def get_valid_int(prompt, min_value, max_value):
    while True:
        try:
            user_input = int(input(prompt))
            if min_value <= user_input <= max_value:
                return user_input
            else:
                print(f"Please provide a number between {min_value} and {max_value}.")
        except ValueError:
            print("Invalid input. Please enter a valid integer.")
```

Screenshot 2: Loading transactions from CSV & function validation

```python
# Brute force method for finding frequent itemsets and generating rules
def brute_force_algorithm(transactions, min_support, min_confidence):
    def count_itemsets(itemsets):
        counts = defaultdict(int)
        for transaction in transactions:
            for itemset in itemsets:
                if set(itemset).issubset(transaction):
                    counts[itemset] += 1
        return counts

    unique_items = set(item for transaction in transactions for item in transaction)
    total_transactions = len(transactions)
    frequent_itemsets = {}
    k = 1

    while True:
        itemsets = list(itertools.combinations(unique_items, k))
        item_counts = count_itemsets(itemsets)
        frequent_items = {frozenset(item): count / total_transactions for item, count in item_counts.items() if count / total_transactions >=
        if not frequent_items:
            break
        frequent_itemsets[k] = frequent_items
        k += 1

    generated_rules = []
    for k in range(2, len(frequent_itemsets) + 1):
        for itemset in frequent_itemsets[k]:
            for i in range(1, k):
                for antecedent in itertools.combinations(itemset, i):
                    antecedent_set = frozenset(antecedent)
                    consequent_set = frozenset(itemset) - antecedent_set
                    if antecedent_set in frequent_itemsets[len(antecedent_set)]:
                        support = frequent_itemsets[k][itemset]
                        confidence = support / frequent_itemsets[len(antecedent_set)][antecedent_set]
                        if confidence >= min_confidence:
                            generated_rules.append((antecedent_set, consequent_set, confidence, support))

    return frequent_itemsets, generated_rules
```

Screenshot 3: Brute Force & Association Rule generation

```python
# Apriori algorithm using the mlxtend library
def apriori_algorithm(transactions, min_support, min_confidence):
    encoder = TransactionEncoder()
    encoded_array = encoder.fit(transactions).transform(transactions)
    df = pd.DataFrame(encoded_array, columns=encoder.columns_)
    frequent_itemsets = apriori_lib(df, min_support=min_support, use_colnames=True)
    rules = association_rules(frequent_itemsets, metric="confidence", min_threshold=min_confidence)
    return frequent_itemsets, rules

# Function to compare results from both algorithms
def compare_algorithms(brute_force_results, apriori_results):
    bf_itemsets, bf_rules = brute_force_results
    ap_itemsets, ap_rules = apriori_results

    print("\nResults Comparison:")
    print(f"Brute Force: {sum(len(itemsets) for itemsets in bf_itemsets.values())} frequent itemsets, {len(bf_rules)} rules")
    print(f"Apriori: {len(ap_itemsets)} frequent itemsets, {len(ap_rules)} rules")

    # Check if both methods yield the same results
    same_itemsets_count = (sum(len(itemsets) for itemsets in bf_itemsets.values()) == len(ap_itemsets))
    same_rules_count = len(bf_rules) == len(ap_rules)
    print(f"Same number of frequent itemsets: {same_itemsets_count}")
    print(f"Same number of rules: {same_rules_count}")
```

Screenshot 4: Apriori & Comparison of all Algorithms

```python
# Main loop for user interaction
while True:
    print("\nWelcome! Please select a store to analyze:")
    for idx, store in enumerate(store_files.keys(), 1):
        print(f"{idx}. {store}")

    # Store selection using integer input validation
    selected_store = get_valid_int("Enter the number of the store you want to analyze (1-5): ", 1, 5)
    store_name = list(store_files.keys())[selected_store - 1]

    try:
        # Load transactions from the selected store's CSV file
        transactions = read_transactions_from_csv(store_files[store_name])

        print(f"\nAnalyzing transactions for {store_name}:")
        min_support = get_valid_float("Enter minimum support (1-100): ", 1, 100)
        min_confidence = get_valid_float("Enter minimum confidence (1-100): ", 1, 100)

        print("\nExecuting algorithms...")

        # Run brute force algorithm
        start_time = time.time()
        bf_itemsets, bf_rules = brute_force_algorithm(transactions, min_support, min_confidence)
        bf_duration = time.time() - start_time

        # Run Apriori algorithm
        start_time = time.time()
        ap_itemsets, ap_rules = apriori_algorithm(transactions, min_support, min_confidence)
        ap_duration = time.time() - start_time

        print(f"\nBrute Force Execution Time: {bf_duration:.4f} seconds")
        print(f"Apriori Execution Time: {ap_duration:.4f} seconds")

        # Determine the faster algorithm
        faster_algorithm = min(("Brute Force", bf_duration), ("Apriori", ap_duration), key=lambda x: x[1])
        print(f"\nThe faster algorithm was: {faster_algorithm[0]} with a duration of {faster_algorithm[1]:.4f} seconds")

        # Compare results and display rules
        compare_algorithms((bf_itemsets, bf_rules), (ap_itemsets, ap_rules))
        show_rules(bf_rules, "Brute Force")
        show_rules(ap_rules, "Apriori")
```

Screenshot 5: Main function for User Interaction

Below screenshots show the output and user interaction:

```
Welcome! Please select a store to analyze:
1. Amazon
2. Best Buy
3. Nike
4. K Mart
5. Target
Enter the number of the store you want to analyze (1-5):  3

Analyzing transactions for Nike:
Enter minimum support (1-100):  55
Enter minimum confidence (1-100):  70
```

Screenshot 6: User selects a store and enters minimum support and confidence for that store.

After entering minimum support and confidence values, we get the following:

```
Executing algorithms...

Brute Force Execution Time: 0.0021 seconds
Apriori Execution Time: 0.0127 seconds

The faster algorithm was: Brute Force with a duration of 0.0021 seconds

Results Comparison:
Brute Force: 8 frequent itemsets, 6 rules
Apriori: 8 frequent itemsets, 6 rules
Same number of frequent itemsets: True
Same number of rules: True

Brute Force Association Rules:
Rule 1: {'Socks'} -> {'Running Shoe'} (Confidence: 0.85, Support: 0.55)
Rule 2: {'Running Shoe'} -> {'Socks'} (Confidence: 0.79, Support: 0.55)
Rule 3: {'Socks'} -> {'Sweatshirts'} (Confidence: 0.92, Support: 0.60)
Rule 4: {'Sweatshirts'} -> {'Socks'} (Confidence: 0.92, Support: 0.60)
Rule 5: {'Running Shoe'} -> {'Sweatshirts'} (Confidence: 0.79, Support: 0.55)
Rule 6: {'Sweatshirts'} -> {'Running Shoe'} (Confidence: 0.85, Support: 0.55)

Apriori Association Rules:
Rule 1: {'Socks'} -> {'Running Shoe'} (Confidence: 0.85, Support: 0.55)
Rule 2: {'Running Shoe'} -> {'Socks'} (Confidence: 0.79, Support: 0.55)
Rule 3: {'Running Shoe'} -> {'Sweatshirts'} (Confidence: 0.79, Support: 0.55)
Rule 4: {'Sweatshirts'} -> {'Running Shoe'} (Confidence: 0.85, Support: 0.55)
Rule 5: {'Socks'} -> {'Sweatshirts'} (Confidence: 0.92, Support: 0.60)
Rule 6: {'Sweatshirts'} -> {'Socks'} (Confidence: 0.92, Support: 0.60)

Item Counts:
Socks: Count = 13, Support = 0.65 (Meets support threshold)
Running Shoe: Count = 14, Support = 0.70 (Meets support threshold)
Modern Pants: Count = 10, Support = 0.50 (Does not meet support threshold)
Sweatshirts: Count = 13, Support = 0.65 (Meets support threshold)
Soccer Shoe: Count = 6, Support = 0.30 (Does not meet support threshold)
Tech Pants: Count = 9, Support = 0.45 (Does not meet support threshold)
Rash Guard: Count = 12, Support = 0.60 (Meets support threshold)
Hoodies: Count = 8, Support = 0.40 (Does not meet support threshold)
Swimming Shirt: Count = 11, Support = 0.55 (Meets support threshold)
Dry Fit V-Nick: Count = 10, Support = 0.50 (Does not meet support threshold)

Would you like to analyze another store? (y/n)
```

Screenshot 7: We get each algorithm time, fastest algorithm, association rules & item count

GitHub Repository

https://github.com/DathwikK/association_rule_mining

References

1. Mlxtend library documentation
2. Brute Force

3. Association Rule Mining
4. ChatGPT
5. Blacbox.ai