Association Rule Mining Project Report

Name: Tejas Belakavadi Kemparaju
NJIT UCID: tb389
Email Address: tb389@njit.edu

Date: 10/10/2024
Professor: Yasser Abduallah
Course: CS 634101 Data Mining
Github Repo: Link

Abstract:

This project implements and evaluates three association rule mining algorithms: Brute Force, Apriori, and FP growth. By applying these algorithms to transactional data from five well-known retailers—Amazon, Best Buy, Nike, Walmart, and Target—we analyze their ability to uncover purchasing patterns and generate meaningful association rules. The project highlights the practical application of data mining techniques in the retail industry, offering insights into the comparative performance of each algorithm, customer purchasing behavior, and the trade-offs in selecting different association rule mining methods.

Introduction:

Association rule mining is a crucial technique in data analysis, particularly useful in sectors like retail and e-commerce. This project aims to implement and compare three major association rule mining algorithms:

- **Brute Force**: A custom-built method that exhaustively evaluates all possible itemsets.
- **Apriori**: A well-known algorithm for frequent itemset generation, implemented using the mlxtend library.
- **FP-Growth**: A tree-based method for mining frequent patterns, also implemented using mlxtend.

These algorithms are tested on transaction datasets from five prominent retail chains—Amazon, Best Buy, Nike, Walmart, and Target. Through this comparative study, we aim to identify each algorithm's relative strengths and limitations in the context of retail data analysis.

Frequent Itemset Discovery:

Frequent set discovery lies at the core of association rule mining, focusing on identifying sets of items that frequently occur together within transactions. This project explores and compares different methods for uncovering these frequent itemsets, ranging from the exhaustive Brute Force approach, which evaluates all

possible combinations, to more efficient algorithms like Apriori and FP-Growth. These comparisons shed light on how various techniques handle the complexity of itemset discovery.

Support and Confidence Metrics: are two key measures guiding the analysis in this project. Support reflects how often an itemset appears within the transaction dataset, providing a foundation for identifying frequent patterns. Confidence measures the likelihood that one item is purchased given the purchase of another, which is crucial for understanding the strength of relationships between items. These metrics are vital for filtering and ranking the association rules, ensuring the discovery of meaningful patterns that can be applied to real-world business scenarios.

The ultimate goal of this project is to generate Association Rules that reveal customer purchasing patterns. These rules provide valuable insights into customer behavior, offering applications for personalized product recommendations and store layout optimization. By analyzing these patterns, businesses can make informed decisions to enhance customer experience and maximize sales opportunities. A critical component of this project is the comparison of Algorithm Efficiency. The computational efficiency of the Brute Force, Apriori, and FP-Growth algorithms is measured through their execution times. This comparison highlights the trade-offs between exhaustive search methods and more optimized approaches, helping to assess which algorithms are best suited for large-scale retail data analysis.

Finally, the project emphasizes the importance of Data Preprocessing in ensuring the success of data mining tasks. The process of loading and preparing transactional data plays a significant role in the overall effectiveness of association rule mining. By illustrating proper preprocessing techniques, this project demonstrates how clean and well-structured data can enhance the performance of the algorithms and lead to more accurate and insightful results.

Project Workflow

Data Preparation and Loading:

- CSV File Creation: The transaction data provided by the professor was expanded and systematically organized into Excel sheets, with separate

sheets created for each store: Amazon, Best Buy, Nike, Walmart, and Target.

- CSV Loading: A custom function, load_transactions_from_csv, was developed to read these CSV files and convert the data into a format that is appropriate for further analysis.

User Interaction: The program includes a user-friendly interface that enables users to select a specific store for analysis and input desired thresholds for support and confidence. Input validation functions are implemented to ensure that all user entries fall within acceptable ranges before proceeding with the analysis.

Algorithm Implementation:

1. **Brute Force Algorithm:**
    a. A custom algorithm is implemented using Python's intertools to generate combinations of items.
    b. This method exhaustively checks all possible itemsets against the defined minimum support threshold.
    c. It generates association rules based on frequent itemsets that meet the minimum confidence requirement.

2. **Apriori Algorithm:**
    a. This approach leverages the Apriori implementation from the mlxtend library.
    b. It efficiently generates frequent itemsets following the apriori principle, which reduces the search space.

3. **FP-Growth Algorithm:** Also implemented using the mlxtend library, the FP-Growth algorithm employs a tree-based structure to efficiently mine frequent patterns from the dataset.

Performance Measurement: Execution time for each algorithm is measured using Python's time module. This allows for direct comparisons of computational efficiency across the Brute Force, Apriori, and FP-Growth algorithms.

Result Comparison and Analysis: The project compares the number of frequent itemsets and association rules generated by each algorithm. It checks for consistency in the results produced by all three methods and identifies the fastest algorithm for each specific analysis.

Rule Display and Interpretation: The project implements functions to display the generated association rules in a clear and readable format. These functions also provide insights into the strength of the discovered associations by reporting their corresponding support and confidence metrics.

Item Analysis: Item counts and their respective support values are calculated and displayed. The analysis also determines whether individual items meet the defined support threshold, helping to assess the relevance of frequent itemsets further.

Screenshots:

amazon_transactions

A Beginner's Guide	**Java: The Complete Reference**	**Java For Dummies**	**Android Programming: The Big Nerd Ranch**	
A Beginner's Guide	Java For Dummies	Java: The Complete Reference		
A Beginner's Guide	Java: The Complete Reference	Java For Dummies	Android Programming: The Big Nerd Ranch	Head First Java 2nd Edition
Android Programming: The Big Nerd Ranch	Beginning Programming with Java	Head First Java 2nd Edition		
Android Programming: The Big Nerd Ranch	Beginning Programming with Java	Java 8 Pocket Guide		
A Beginner's Guide	Android Programming: The Big Nerd Ranch	Head First Java 2nd Edition		
A Beginner's Guide	Head First Java 2nd Edition	Beginning Programming with Java		
Java: The Complete Reference	Java For Dummies	Android Programming: The Big Nerd Ranch		
Java For Dummies	Android Programming: The Big Nerd Ranch	Head First Java 2nd Edition	Beginning Programming with Java	
Beginning Programming with Java	Java 8 Pocket Guide	C++ Programming in Easy Steps		
A Beginner's Guide	Java: The Complete Reference	Java For Dummies	Android Programming: The Big Nerd Ranch	
A Beginner's Guide	Java: The Complete Reference	Java For Dummies	HTML and CSS: Design and Build Websites	
A Beginner's Guide	Java: The Complete Reference	Java For Dummies	Java 8 Pocket Guide	HTML and CSS: Design and Build Websites
Java For Dummies	Android Programming: The Big Nerd Ranch	Head First Java 2nd Edition		
Java For Dummies	Android Programming: The Big Nerd Ranch			
A Beginner's Guide	Java: The Complete Reference	Java For Dummies	Android Programming: The Big Nerd Ranch	
A Beginner's Guide	Java: The Complete Reference	Java For Dummies	Android Programming: The Big Nerd Ranch	
Head First Java 2nd Edition	Beginning Programming with Java	Java 8 Pocket Guide		
Android Programming: The Big Nerd Ranch	Head First Java 2nd Edition			
A Beginner's Guide	Java: The Complete Reference	Java For Dummies		

Figure 1: Amazon Transactions

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

# Function to load transactions from CSV
def load_transactions_from_csv(filename):
    df = pd.read_csv(filename, header=None)
    return [set(str(item) for item in transaction if pd.notna(item)) for transaction in df.values.tolist()]

# Function to validate input for support and confidence values
def validate_input_float(prompt, min_val, max_val):
    while True:
        try:
            value = float(input(prompt))
            if min_val <= value <= max_val:
                return value / 100  # Convert percentage to decimal
            else:
                print(f"Please enter a value between {min_val} and {max_val}.")
        except ValueError:
            print("Invalid input. Please enter a number.")

# Function to validate input for store selection
def validate_input_int(prompt, min_val, max_val):
    while True:
        try:
            value = int(input(prompt))
            if min_val <= value <= max_val:
                return value
            else:
                print(f"Please enter a number between {min_val} and {max_val}.")
        except ValueError:
            print("Invalid input. Please enter an integer.")
```

```python
# Store items for transaction generation
stores = {
    "Amazon": "amazon_transactions.csv",
    "Best Buy": "best buy_transactions.csv",
    "Nike": "nike_transactions.csv",
    "Walmart": "walmart_transactions.csv",
    "Target": "target_transactions.csv"
}

# Brute force algorithm for finding frequent itemsets and generating rules
def brute_force(transactions, min_support, min_confidence):
    def get_item_counts(itemsets):
        item_counts = defaultdict(int)
        for transaction in transactions:
            for itemset in itemsets:
                if set(itemset).issubset(transaction):
                    item_counts[itemset] += 1
        return item_counts

    items = set(item for transaction in transactions for item in transaction)
    n = len(transactions)
    frequent_itemsets = {}
    k = 1

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

    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 = frozenset(antecedent)
```

Figure 2: Importing transaction from CSV

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

    items = set(item for transaction in transactions for item in transaction)
    n = len(transactions)
    frequent_itemsets = {}
    k = 1

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

    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 = frozenset(antecedent)
                    consequent = frozenset(itemset) - antecedent
                    if antecedent in frequent_itemsets[len(antecedent)]:
                        support = frequent_itemsets[k][itemset]
                        confidence = support / frequent_itemsets[len(antecedent)][antecedent]
                        if confidence >= min_confidence:
                            rules.append((antecedent, consequent, confidence, support))

    return frequent_itemsets, rules
```

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

# FP-Growth algorithm using the mlxtend library
def fp_growth_method(transactions, min_support, min_confidence):
    te = TransactionEncoder()
    te_ary = te.fit(transactions).transform(transactions)
    df = pd.DataFrame(te_ary, columns=te.columns_)
    frequent_itemsets = fpgrowth(df, min_support=min_support, use_colnames=True)
    rules = association_rules(frequent_itemsets, metric="confidence", min_threshold=min_confidence)
    return frequent_itemsets, rules

# Compare the results of brute force, Apriori, and FP-Growth algorithms
def compare_results(brute_force_results, apriori_results, fpgrowth_results):
    bf_itemsets, bf_rules = brute_force_results
    ap_itemsets, ap_rules = apriori_results
    fp_itemsets, fp_rules = fpgrowth_results

    print("\nComparison of results:")
    print(f"Brute Force: {sum(len(itemsets) for itemsets in bf_itemsets.values())} frequent itemsets, {len(bf_rules)
    print(f"Apriori: {len(ap_itemsets)} frequent itemsets, {len(ap_rules)} rules")
    print(f"FP-Growth: {len(fp_itemsets)} frequent itemsets, {len(fp_rules)} rules")

    # Check if all algorithms produce the same results
    same_itemsets = (sum(len(itemsets) for itemsets in bf_itemsets.values()) == len(ap_itemsets) == len(fp_itemsets)
    same_rules = len(bf_rules) == len(ap_rules) == len(fp_rules)
    print(f"Same number of frequent itemsets: {same_itemsets}")
    print(f"Same number of rules: {same_rules}")
```

Figure 3: Implementing Brute Force and Apriori Algorithm

```
Welcome User! Here are the available stores you can select from:
1. Amazon
2. Best Buy
3. Nike
4. Walmart
5. Target
Enter the number of the store you want to analyze (1-5): 1

Analyzing Amazon transactions:
Enter minimum support (1-100): 30
Enter minimum confidence (1-100): 56

Running algorithms...

Brute Force Time: 0.0016 seconds
Apriori Time: 0.0075 seconds
FP-Growth Time: 0.0029 seconds
```

Figure 4: User selects a store and enters minimum support and confidence

```
The fastest algorithm was: Brute Force with a time of 0.0016 seconds

Comparison of results:
Brute Force: 15 frequent itemsets, 20 rules
Apriori: 15 frequent itemsets, 20 rules
FP-Growth: 15 frequent itemsets, 20 rules
Same number of frequent itemsets: True
Same number of rules: True

Brute Force Association Rules:
Rule 1: {"A Beginner's Guide"} -> {'Java: The Complete Reference'} (Confidence: 0.82, Support: 0.45)
Rule 2: {'Java: The Complete Reference'} -> {"A Beginner's Guide"} (Confidence: 0.90, Support: 0.45)
Rule 3: {'Java For Dummies'} -> {"A Beginner's Guide"} (Confidence: 0.69, Support: 0.45)
Rule 4: {"A Beginner's Guide"} -> {'Java For Dummies'} (Confidence: 0.82, Support: 0.45)
Rule 5: {'Java For Dummies'} -> {'Java: The Complete Reference'} (Confidence: 0.77, Support: 0.50)
Rule 6: {'Java: The Complete Reference'} -> {'Java For Dummies'} (Confidence: 1.00, Support: 0.50)
Rule 7: {'Java: The Complete Reference'} -> {'Android Programming: The Big Nerd Ranch'} (Confidence: 0.60, Support:
0.30)

Rule 8: {'Java For Dummies'} -> {'Android Programming: The Big Nerd Ranch'} (Confidence: 0.69, Support: 0.45)
Rule 9: {'Android Programming: The Big Nerd Ranch'} -> {'Java For Dummies'} (Confidence: 0.69, Support: 0.45)
Rule 10: {'Head First Java 2nd Edition'} -> {'Android Programming: The Big Nerd Ranch'} (Confidence: 0.75, Support:
0.30)
Rule 11: {'Java For Dummies'} -> {"A Beginner's Guide", 'Java: The Complete Reference'} (Confidence: 0.69, Support:
0.45)
Rule 12: {"A Beginner's Guide"} -> {'Java For Dummies', 'Java: The Complete Reference'} (Confidence: 0.82, Support:
0.45)
Rule 13: {'Java: The Complete Reference'} -> {'Java For Dummies', "A Beginner's Guide"} (Confidence: 0.90, Support:
0.45)
Rule 14: {'Java For Dummies', "A Beginner's Guide"} -> {'Java: The Complete Reference'} (Confidence: 1.00, Support:
0.45)
Rule 15: {'Java For Dummies', 'Java: The Complete Reference'} -> {"A Beginner's Guide"} (Confidence: 0.90, Support:
0.45)
Rule 16: {"A Beginner's Guide", 'Java: The Complete Reference'} -> {'Java For Dummies'} (Confidence: 1.00, Support:
0.45)
Rule 17: {'Java: The Complete Reference'} -> {'Java For Dummies', 'Android Programming: The Big Nerd Ranch'} (Confi
dence: 0.60, Support: 0.30)

Rule 18: {'Java For Dummies', 'Android Programming: The Big Nerd Ranch'} -> {'Java: The Complete Reference'} (Confi
dence: 0.67, Support: 0.30)
Rule 19: {'Java: The Complete Reference', 'Android Programming: The Big Nerd Ranch'} -> {'Java For Dummies'} (Confi
dence: 1.00, Support: 0.30)
Rule 20: {'Java For Dummies', 'Java: The Complete Reference'} -> {'Android Programming: The Big Nerd Ranch'} (Confi
dence: 0.60, Support: 0.30)

Apriori Association Rules:
Rule 1: {'Java For Dummies'} -> {"A Beginner's Guide"} (Confidence: 0.69, Support: 0.45)
Rule 2: {"A Beginner's Guide"} -> {'Java For Dummies'} (Confidence: 0.82, Support: 0.45)
Rule 3: {"A Beginner's Guide"} -> {'Java: The Complete Reference'} (Confidence: 0.82, Support: 0.45)
Rule 4: {'Java: The Complete Reference'} -> {"A Beginner's Guide"} (Confidence: 0.90, Support: 0.45)
Rule 5: {'Head First Java 2nd Edition'} -> {'Android Programming: The Big Nerd Ranch'} (Confidence: 0.75, Support:
0.30)
```

```
Rule 6: {'Java For Dummies'} -> {'Android Programming: The Big Nerd Ranch'} (Confidence: 0.69, Support: 0.45)
Rule 7: {'Android Programming: The Big Nerd Ranch'} -> {'Java For Dummies'} (Confidence: 0.69, Support: 0.45)
Rule 8: {'Java: The Complete Reference'} -> {'Android Programming: The Big Nerd Ranch'} (Confidence: 0.60, Support:
0.30)
Rule 9: {'Java For Dummies'} -> {'Java: The Complete Reference'} (Confidence: 0.77, Support: 0.50)
Rule 10: {'Java: The Complete Reference'} -> {'Java For Dummies'} (Confidence: 1.00, Support: 0.50)
Rule 11: {'Java For Dummies', "A Beginner's Guide"} -> {'Java: The Complete Reference'} (Confidence: 1.00, Support:
0.45)
Rule 12: {'Java For Dummies', 'Java: The Complete Reference'} -> {"A Beginner's Guide"} (Confidence: 0.90, Support:
0.45)
Rule 13: {"A Beginner's Guide", 'Java: The Complete Reference'} -> {'Java For Dummies'} (Confidence: 1.00, Support:
0.45)
Rule 14: {'Java For Dummies'} -> {"A Beginner's Guide", 'Java: The Complete Reference'} (Confidence: 0.69, Support:
0.45)
Rule 15: {"A Beginner's Guide"} -> {'Java For Dummies', 'Java: The Complete Reference'} (Confidence: 0.82, Support:
0.45)
Rule 16: {'Java: The Complete Reference'} -> {'Java For Dummies', "A Beginner's Guide"} (Confidence: 0.90, Support:
0.45)

Rule 17: {'Java For Dummies', 'Android Programming: The Big Nerd Ranch'} -> {'Java: The Complete Reference'} (Confi
dence: 0.67, Support: 0.30)
Rule 18: {'Android Programming: The Big Nerd Ranch', 'Java: The Complete Reference'} -> {'Java For Dummies'} (Confi
dence: 1.00, Support: 0.30)
Rule 19: {'Java For Dummies', 'Java: The Complete Reference'} -> {'Android Programming: The Big Nerd Ranch'} (Confi
dence: 0.60, Support: 0.30)
Rule 20: {'Java: The Complete Reference'} -> {'Java For Dummies', 'Android Programming: The Big Nerd Ranch'} (Confi
dence: 0.60, Support: 0.30)

FP-Growth Association Rules:
Rule 1: {'Java For Dummies'} -> {'Android Programming: The Big Nerd Ranch'} (Confidence: 0.69, Support: 0.45)
Rule 2: {'Android Programming: The Big Nerd Ranch'} -> {'Java For Dummies'} (Confidence: 0.69, Support: 0.45)
Rule 3: {'Java For Dummies'} -> {"A Beginner's Guide"} (Confidence: 0.69, Support: 0.45)
Rule 4: {"A Beginner's Guide"} -> {'Java For Dummies'} (Confidence: 0.82, Support: 0.45)
Rule 5: {'Java For Dummies'} -> {'Java: The Complete Reference'} (Confidence: 0.77, Support: 0.50)
Rule 6: {'Java: The Complete Reference'} -> {'Java For Dummies'} (Confidence: 1.00, Support: 0.50)
Rule 7: {"A Beginner's Guide"} -> {'Java: The Complete Reference'} (Confidence: 0.82, Support: 0.45)
Rule 8: {'Java: The Complete Reference'} -> {"A Beginner's Guide"} (Confidence: 0.90, Support: 0.45)

Rule 9: {'Java: The Complete Reference'} -> {'Android Programming: The Big Nerd Ranch'} (Confidence: 0.60, Support:
0.30)
Rule 10: {'Java For Dummies', "A Beginner's Guide"} -> {'Java: The Complete Reference'} (Confidence: 1.00, Support:
0.45)
Rule 11: {'Java For Dummies', 'Java: The Complete Reference'} -> {"A Beginner's Guide"} (Confidence: 0.90, Support:
0.45)
Rule 12: {"A Beginner's Guide", 'Java: The Complete Reference'} -> {'Java For Dummies'} (Confidence: 1.00, Support:
0.45)
Rule 13: {'Java For Dummies'} -> {"A Beginner's Guide", 'Java: The Complete Reference'} (Confidence: 0.69, Support:
0.45)
Rule 14: {"A Beginner's Guide"} -> {'Java For Dummies', 'Java: The Complete Reference'} (Confidence: 0.82, Support:
0.45)
Rule 15: {'Java: The Complete Reference'} -> {'Java For Dummies', "A Beginner's Guide"} (Confidence: 0.90, Support:
0.45)
Rule 16: {'Java For Dummies', 'Android Programming: The Big Nerd Ranch'} -> {'Java: The Complete Reference'} (Confi
dence: 0.67, Support: 0.30)
Rule 17: {'Android Programming: The Big Nerd Ranch', 'Java: The Complete Reference'} -> {'Java For Dummies'} (Confi
dence: 1.00, Support: 0.30)
Rule 18: {'Java For Dummies', 'Java: The Complete Reference'} -> {'Android Programming: The Big Nerd Ranch'} (Confi
dence: 0.60, Support: 0.30)
Rule 19: {'Java: The Complete Reference'} -> {'Java For Dummies', 'Android Programming: The Big Nerd Ranch'} (Confi
dence: 0.60, Support: 0.30)
Rule 20: {'Head First Java 2nd Edition'} -> {'Android Programming: The Big Nerd Ranch'} (Confidence: 0.75, Support:
0.30)

Item Counts:
Android Programming: The Big Nerd Ranch: Count = 13, Support = 0.65 (Meets support threshold)
Java For Dummies: Count = 13, Support = 0.65 (Meets support threshold)
A Beginner's Guide: Count = 11, Support = 0.55 (Meets support threshold)
Java: The Complete Reference: Count = 10, Support = 0.50 (Meets support threshold)
Head First Java 2nd Edition: Count = 8, Support = 0.40 (Meets support threshold)
Beginning Programming with Java: Count = 6, Support = 0.30 (Meets support threshold)
Java 8 Pocket Guide: Count = 4, Support = 0.20 (Does not meet support threshold)
C++ Programming in Easy Steps: Count = 1, Support = 0.05 (Does not meet support threshold)
HTML and CSS: Design and Build Websites: Count = 2, Support = 0.10 (Does not meet support threshold)

Do you want to analyze another store? (y/n)

y
```

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

```
Welcome User! Here are the available stores you can select from:
1. Amazon
2. Best Buy
3. Nike
4. Walmart
5. Target
Enter the number of the store you want to analyze (1-5): 1
```

References:

Github: https://github.com/tejas100/Tejas_Belakavadi_Kemparaju_DM_midtermproject

References

1. Mlxtend library documentation
2. Brute Force
3. ChatGPT