NAME: Aesha Dobariya
NJIT UCID: ad2389
Email Address: ad2389@njit.edu
13th October 2024
Professor: Yasser Abduallah
CS 634 - Data Mining

Midterm Project Report

Applying the Apriori Algorithm to Retail Data Analysis

Abstract:

This project delves into the Apriori Algorithm, a key data mining technique, to reveal patterns in retail sales data. By coding the algorithm and using various data mining methods, I test its performance and usefulness. I also build custom data mining tools to create a specialized model for extracting valuable insights from sales records.

Introduction:

Data mining helps uncover hidden trends in large datasets. Our project centers on the Apriori Algorithm, a well-known method for finding links between items, and how it works with retail data. We'll cover the main data mining ideas and principles used in our work.

Creating and showing association rules is central to the Apriori Algorithm. To make these rules, I needed to find the most common items in the list of sales. Once I found these frequent items, I had to figure out their support value based on the user's chosen support level. After calculating support for each item, we can remove items that don't meet the user's support threshold. The Apriori algorithm is a classic data mining method that uses a thorough approach to find common item groups and make association rules. It works by slowly increasing the size of item groups and removing those that don't appear often enough.

In this project, I used the Apriori algorithm on a custom retail store dataset to find frequent item groups and association rules. The main steps were:

Setting up dictionaries for possible and frequent item groups.

Reading the dataset and item lists from CSV files.

Cleaning up the dataset to make sure items are in order and not repeated.

Getting user input for minimum support and confidence levels.

Repeatedly making possible item groups and updating frequent ones using the Apriori algorithm, which looks at all possible item combinations.

Key Concepts and Principles:

Finding Frequent Item Groups:

The Apriori Algorithm focuses on finding groups of items that often appear together in sales. These groups show us what customers tend to buy together.

Support and Confidence:

Two important measures in data mining are support and confidence. Support tells us how often an item or group of items appears, while confidence shows how likely items are to be bought together. We use these measures to guide our analysis.

Association Rules:

By identifying strong associations between items, I uncover patterns in customer purchasing behavior. These insights are crucial for enhancing sales strategies, particularly in product recommendations.

Project Workflow:

Our project follows a systematic approach, incorporating various stages and the implementation of the Apriori Algorithm:

Data Loading and Preprocessing:

We start by importing transaction data from a retail store dataset. Each transaction represents a customer's purchase. To ensure data quality, we clean the dataset, eliminating duplicates and arranging items in a predefined order.

Setting Minimum Support and Confidence:

User input is vital in data mining. We gather the user's preferred minimum support and confidence levels to filter out less relevant patterns.

Iterating Through Candidate Itemsets:

The Apriori Algorithm is applied iteratively to generate candidate itemsets of increasing sizes. We begin with single items (K = 1) and progress to K = 2, K = 3, and so on. This process involves a comprehensive method of creating all possible itemset combinations.

Calculating Support Count:

For each candidate itemset, we determine its support by counting its occurrence in transactions. Itemsets meeting the minimum support threshold are retained, while others are discarded.

Determining Confidence:

We assess the confidence of association rules, indicating the strength of item relationships. This step requires careful comparison of support values for individual items and itemsets.

Generating Association Rules:

We extract association rules that meet both minimum support and confidence criteria. These rules provide valuable insights into frequently co-purchased items.

Results and Evaluation:

We evaluate the project's effectiveness based on performance metrics such as support, confidence, and the resulting association rules. We also compare our custom Apriori Algorithm implementation with the Apriori library to assess its reliability.

Conclusion:

In summary, our project showcases the application of data mining concepts and methods. We successfully implemented the Apriori Algorithm to extract meaningful association rules from retail transaction data. The comprehensive approach, custom algorithm design, and adherence to user-defined parameters demonstrate the power of data mining in uncovering valuable patterns for retail decision-making.

To demonstrate the program's functionality, I've included screenshots of its execution in the Terminal.

Screenshots:

```
[3]: import pandas as pd
     df = pd.read_csv(r"C:\Users\DELL\Downloads\amazon.csv", delimiter=';')
     print(df)
     df
```

```
                                  Items
0                  Toothpaste,Detergent
1                           Coffee,Soap
2                        Detergent,Soap
3                 Diapers,Milk,Eggs,Cereal
4         Milk,Toothpaste,Soap,Eggs,Cereal
5   Detergent,Toothpaste,Shampoo,Soap,Coffee
6                       Bread,Detergent
7                     Detergent,Diapers
8             Toothpaste,Soap,Eggs,Bread
9                      Milk,Cereal,Eggs
10      Cereal,Detergent,Coffee,Eggs,Shampoo
11                   Coffee,Cereal,Soap
12          Toothpaste,Shampoo,Soap,Bread
13          Milk,Cereal,Shampoo,Eggs,Bread
14                    Soap,Diapers,Milk
15                     Cereal,Detergent
16                      Milk,Toothpaste
17   Shampoo,Cereal,Diapers,Coffee,Detergent
18                Coffee,Diapers,Shampoo
19              Shampoo,Soap,Milk,Diapers
```

```
[4]: import pandas as pd
     df = pd.read_csv(r"C:\Users\DELL\Downloads\Kmart.csv", delimiter=';')
     print(df)
     df
```

```
                                Diapers,Milk
0                             Detergent,Milk
1                            Bread,Milk,Eggs
2               Bread,Shampoo,Coffee,Detergent
3                        Eggs,Detergent,Soap
4              Milk,Soap,Cereal,Diapers,Detergent
5                          Detergent,Shampoo
6                  Diapers,Eggs,Shampoo,Milk
7                             Shampoo,Coffee
8                          Detergent,Shampoo
9                                Cereal,Milk
10  Shampoo,Detergent,Toothpaste,Coffee,Cereal
11                         Toothpaste,Coffee
12              Eggs,Shampoo,Detergent,Milk
13              Toothpaste,Detergent,Coffee
14                  Cereal,Diapers,Detergent
15        Diapers,Toothpaste,Bread,Soap,Eggs
16            Cereal,Toothpaste,Shampoo,Eggs
17                  Shampoo,Soap,Bread,Eggs
18             Diapers,Milk,Cereal,Eggs,Soap
```

```
[5]: import pandas as pd
     df = pd.read_csv(r"C:\Users\DELL\Downloads\BestBuy.csv", delimiter=';')
     print(df)
     df
```

```
                          Cereal,Eggs
0                   Milk,Bread,Coffee
1   Shampoo,Eggs,Diapers,Detergent,Coffee
2      Soap,Milk,Coffee,Detergent,Shampoo
3              Soap,Milk,Bread,Toothpaste
4       Cereal,Milk,Bread,Toothpaste,Eggs
5        Detergent,Bread,Toothpaste,Soap
6                        Soap,Eggs,Bread
7              Soap,Milk,Coffee,Detergent
8            Milk,Cereal,Toothpaste,Bread
9                         Cereal,Shampoo
10                             Soap,Eggs
11      Milk,Diapers,Detergent,Toothpaste
12                    Toothpaste,Shampoo
13                      Coffee,Eggs,Milk
14             Cereal,Diapers,Eggs,Coffee
15          Milk,Detergent,Diapers,Coffee
16                        Detergent,Milk
17          Milk,Detergent,Cereal,Shampoo
18                     Eggs,Cereal,Bread
```

```
[6]: import pandas as pd
     df = pd.read_csv(r"C:\Users\DELL\Downloads\Nike.csv", delimiter=';')
     print(df)
     df
```

```
                              Cereal,Eggs,Bread
0                                Detergent,Soap
1                          Bread,Cereal,Shampoo
2          Eggs,Diapers,Detergent,Milk,Shampoo
3                              Cereal,Milk,Eggs
4                             Coffee,Toothpaste
5         Soap,Bread,Shampoo,Coffee,Toothpaste
6                       Bread,Soap,Shampoo,Eggs
7                  Eggs,Diapers,Detergent,Soap
8                          Diapers,Coffee,Bread
9                             Toothpaste,Cereal
10                            Diapers,Detergent
11   Shampoo,Toothpaste,Detergent,Soap,Bread
12              Toothpaste,Bread,Soap,Diapers
13                 Bread,Detergent,Coffee,Soap
14  Toothpaste,Shampoo,Detergent,Soap,Coffee
15                                Cereal,Bread
16                             Shampoo,Diapers
17                                 Milk,Cereal
18                   Eggs,Coffee,Cereal,Diapers
```

```
import pandas as pd
from itertools import combinations
from mlxtend.frequent_patterns import apriori, association_rules, fpgrowth
from mlxtend.preprocessing import TransactionEncoder
import time
```

Fig 1: Libraries used

```python
import os
import csv
import pandas as pd
import time
from itertools import combinations
from mlxtend.frequent_patterns import apriori, association_rules, fpgrowth
from mlxtend.preprocessing import TransactionEncoder

file_paths = {
    "Amazon": r"C:\Users\DELL\Downloads\amazon.csv",
    "BestBuy": r"C:\Users\DELL\Downloads\BestBuy.csv",
    "KMart": r"C:\Users\DELL\Downloads\Kmart.csv",
    "Nike": r"C:\Users\DELL\Downloads\Nike.csv"
}

# Extract transactions from CSV file
def load_transactions(file_path):
    with open(file_path, newline='') as csvfile:
        reader = csv.reader(csvfile)
        transactions = [list(filter(None, row)) for row in reader]  # Filter out empty items in rows
    return transactions

# Applying Brute force method to generate frequent items
def generate_frequent_itemsets(transactions, support_threshold):
    item_count = {}
    for transaction in transactions:
        for item in transaction:
            item_count[item] = item_count.get(item, 0) + 1

    frequent_itemsets = {1: {item: count for item, count in item_count.items() if count / len(transactions) >= support_threshold}}

    k = 2
    while True:
        prev_itemsets = list(frequent_itemsets[k - 1].keys())
        new_itemsets = list(combinations(prev_itemsets, k))
        item_count = {}
        for transaction in transactions:
            transaction_set = set(transaction)
            for itemset in new_itemsets:
                if set(itemset).issubset(transaction_set):
                    item_count[itemset] = item_count.get(itemset, 0) + 1

        frequent_itemsets[k] = {itemset: count for itemset, count in item_count.items() if count / len(transactions) >= support_threshold}
        if not frequent_itemsets[k]:
            del frequent_itemsets[k]
            break
        k += 1
    return frequent_itemsets
```

Fig 2: Reading csv files and generating frequent items

```python
# Applying Apriori Algorithm
def apriori_algorithm(transactions, support_threshold, confidence_threshold):
    te = TransactionEncoder()
    te_ary = te.fit(transactions).transform(transactions)
    df = pd.DataFrame(te_ary, columns=te.columns_)

    frequent_itemsets = apriori(df, min_support=support_threshold, use_colnames=True)
    rules = association_rules(frequent_itemsets, metric="confidence", min_threshold=confidence_threshold)

    return frequent_itemsets, rules

# Applying FP-Growth Algorithm
def fpgrowth_algorithm(transactions, support_threshold, confidence_threshold):
    te = TransactionEncoder()
    te_ary = te.fit(transactions).transform(transactions)
    df = pd.DataFrame(te_ary, columns=te.columns_)

    frequent_itemsets = fpgrowth(df, min_support=support_threshold, use_colnames=True)
    rules = association_rules(frequent_itemsets, metric="confidence", min_threshold=confidence_threshold)

    return frequent_itemsets, rules
```

Fig 3: Applying apriori and fp growth algorithm

```python
# Comparing by timing function
def measure_execution_time(algorithm_func, *args):
    start_time = time.time()
    result = algorithm_func(*args)
    end_time = time.time()
    return result, end_time - start_time
```

Fig 4: Comparing time function

```python
# Source code
while True:
    # user-defined entry or exit
    print("\nAvailable databases:")
    for i, name in enumerate(file_paths.keys(), 1):
        print(f"{i}. {name}")
    print("0. Exit")

    choice = int(input("Enter the number corresponding to the database you'd like to choose (or 0 to exit): "))

    # Exit the loop if the user chooses 0
    if choice == 0:
        print("Exiting the program.")
        break

    #selected database
    db_name = list(file_paths.keys())[choice - 1]

    # Load the selected transactions
    transactions = load_transactions(file_paths[db_name])
    print(f"Loaded {len(transactions)} transactions from {db_name}.")

    # User-defined for support and confidence thresholds
    support_threshold = float(input("Enter support threshold in % (e.g., 10 for 10%): ")) / 100
    confidence_threshold = float(input("Enter confidence threshold in % (e.g., 20 for 20%): ")) / 100

    print(f"\nProcessing {db_name} with support {support_threshold * 100}% and confidence {confidence_threshold * 100}%...")

    # Brute Force
    bf_result, bf_time = measure_execution_time(generate_frequent_itemsets, transactions, support_threshold)
    print(f"\nBrute Force Frequent Itemsets:\n{bf_result}")
    print(f"Brute Force Time: {bf_time:.4f}s")

    # Apriori
    apriori_result, apriori_time = measure_execution_time(apriori_algorithm, transactions, support_threshold, confidence_threshold)
    print(f"\nApriori Frequent Itemsets:\n{apriori_result[0]}")
    print(f"Apriori Rules:\n{apriori_result[1]}")
    print(f"Apriori Time: {apriori_time:.4f}s")

    # FP-Growth
    fp_result, fp_time = measure_execution_time(fpgrowth_algorithm, transactions, support_threshold, confidence_threshold)
    print(f"\nFP-Growth Frequent Itemsets:\n{fp_result[0]}")
    print(f"FP-Growth Rules:\n{fp_result[1]}")
    print(f"FP-Growth Time: {fp_time:.4f}s")

    #if user wants to analyze different dataset
    continue_choice = input("\nDo you want to analyze another dataset? (yes/no): ").strip().lower()
    if continue_choice != 'yes':
        print("Exiting the program.")
        break
```

Fig 5: Main source code

Output:

```
Available databases:
1. Amazon
2. BestBuy
3. KMart
4. Nike
0. Exit
Enter the number corresponding to the database you'd like to choose (or 0 to exit):  1
Loaded 21 transactions from Amazon.
Enter support threshold in % (e.g., 10 for 10%):  20
Enter confidence threshold in % (e.g., 20 for 20%):  60

Processing Amazon with support 20.0% and confidence 60.0%...
```

Fig 1: User Defined datasets

```
Brute Force Frequent Itemsets:
{1: {'Toothpaste': 6, 'Detergent': 8, 'Coffee': 6, 'Soap': 9, 'Diapers': 6, 'Milk': 7, 'Eggs': 6, 'Cereal': 8, 'Shampoo': 7}, 2: {('Eggs', 'Cereal'): 5}}
Brute Force Time: 0.0000s
```

Fig 2: Brute Force time

```
Apriori Frequent Itemsets:
    support       itemsets
0  0.380952       (Cereal)
1  0.285714       (Coffee)
2  0.380952    (Detergent)
3  0.285714      (Diapers)
4  0.285714         (Eggs)
5  0.333333         (Milk)
6  0.333333      (Shampoo)
7  0.428571         (Soap)
8  0.285714   (Toothpaste)
9  0.238095 (Eggs, Cereal)
Apriori Rules:
  antecedents consequents  antecedent support  consequent support   support  \
0      (Eggs)    (Cereal)            0.285714            0.380952  0.238095
1    (Cereal)      (Eggs)            0.380952            0.285714  0.238095

   confidence    lift  leverage  conviction  zhangs_metric
0    0.833333  2.1875  0.129252    3.714286       0.760000
1    0.625000  2.1875  0.129252    1.904762       0.876923
Apriori Time: 0.0430s
```

Fig 3: Apriori algorithm and rules

```
FP-Growth Frequent Itemsets:
    support       itemsets
0  0.380952    (Detergent)
1  0.285714   (Toothpaste)
2  0.428571         (Soap)
3  0.285714       (Coffee)
4  0.380952       (Cereal)
5  0.333333         (Milk)
6  0.285714         (Eggs)
7  0.285714      (Diapers)
8  0.333333      (Shampoo)
9  0.238095 (Eggs, Cereal)
FP-Growth Rules:
  antecedents consequents  antecedent support  consequent support   support  \
0      (Eggs)    (Cereal)            0.285714            0.380952  0.238095
1    (Cereal)      (Eggs)            0.380952            0.285714  0.238095

   confidence    lift  leverage  conviction  zhangs_metric
0    0.833333  2.1875  0.129252    3.714286       0.760000
1    0.625000  2.1875  0.129252    1.904762       0.876923
FP-Growth Time: 0.0036s
```

Fig 4: FP-Growth algorithm and rules

Other:

The source code (.py file) and data sets (.csv files) will be attached to the zip file.

Link to Git Repository:

https://github.com/aesha2492/DM_midtermproj