Custom Data Mining Project Report: Apriori Algorithm Implementation

Name: Vignesh Surya Mantha

NJIT UCID: vm675

Email Address: vm675@njit.edu

Date: 8th October

Professor: Yasser Abduallah

Course: CS 634 Data Mining (101)

Abstract

This project delves into the implementation of the Apriori Algorithm, one of the key algorithms in data mining, to identify patterns and associations within retail transaction datasets. The aim of this project was to extract frequent itemsets and generate association rules to compare the performance of different algorithms such as Brute Force, Apriori, and FP-Tree in terms of mining data and discovering valuable insights.

Introduction

In today's data driven world finding hidden patterns in large datasets is crucial for businesses, one key technique used in data mining is the Apriori Algorithm which helps uncover relationships between items purchased together by customers. through this project I applied the Apriori Algorithm to a retail dataset to find frequent itemsets and association rules helping me better understand customer behavior.

Tools and Libraries Used

- **pandas**: Used for data manipulation, reading transactions from CSV files, and processing data.
- **itertools**: Used to generate combinations of items for the Brute Force method.
- **mlxtend**: Provides functions for Apriori algorithm implementation and association rule generation.
- **pyfpgrowth**: Used to implement FP-Tree algorithm for discovering frequent patterns and association rules.
- **time**: Utilized for measuring the performance of the algorithms.

Core Concepts and Principles

The key idea behind the Apriori Algorithm is to identify frequent itemsets that appear together in transactions helping for businesses make informed decisions about which products to promote together or how to optimize their inventory.

Support and Confidence

Support is the measure of how often an item or itemset appears in transactions while confidence measures how likely it is that items are purchased together. In this project I experimented with various support and confidence thresholds observing how adjusting these values changes the results and rules generated.

Association Rules

By determining strong association rules I was able to identify which products tend to be purchased together by customers. for example, if customers buy a laptop - they may also purchase a laptop case or mouse so these insights can be invaluable for creating marketing strategies or bundling offers.

Project Workflow

I followed a structured approach to analyze the dataset using the Apriori Algorithm, Brute Force method, and FP-Tree algorithm.

Here's a breakdown of the workflow I followed:

Data Loading and Preprocessing

The first step in my project was to load transaction data from CSV files for different stores (e.x - Grocery, Clothing, Electronics). Each file contained several transactions, each listing the products bought together. I had to preprocess the data by removing missing or irrelevant values (like NaN – not a number entries).

Determination of Minimum Support and Confidence

I collected user input for the minimum support and confidence levels, these parameters helped filter the itemsets and association rules that were deemed important. higher support values meaning that only itemsets appearing frequently in transactions would be considered.

Generating Itemsets and Rules with Apriori Algorithm

Using the Apriori Algorithm, i iteratively generated candidate itemsets starting from single items and progressively increasing the number of items. each itemset was evaluated based on its support and while association rules were generated for itemsets that met the minimum confidence threshold -> The process revealed interesting patterns such as which pharmacy items are frequently purchased together.

Comparison with FP-Tree Algorithm

After generating rules with Apriori i also implemented the FP-Tree algorithm. This algorithm uses a more efficient tree-based approach to generate frequent patterns and I compared its performance with Apriori. The FP-Tree algorithm performed faster in some cases due to its compact tree structure but both algorithms yielded similar rules.

Results and Evaluation

The project revealed meaningful insights from the retail dataset. For instance, association rules showed that customers who bought a laptop were likely to also buy a mouse, and those who purchased gym clothes also often bought water bottles. These insights were extracted through the iterative application of the Apriori and FP-Tree algorithms.

I also measured the time taken by each algorithm to process the dataset and generate the rules. As expected the FP-Tree algorithm proved faster, especially with larger datasets, while the Brute Force method took significantly longer. However both Apriori and FP-Tree yielded similar frequent itemsets and rules.

Conclusion Demonstration Store Selection:

Through this project, I explored the power of data mining algorithms like Apriori, Brute Force, and FP-Tree to uncover associations within retail data. The custom implementation of the Apriori Algorithm revealed valuable patterns that can inform business decisions in a retail context, while comparisons with other algorithms highlighted the advantages of different approaches. Overall, this project emphasized the importance of data mining in understanding consumer behavior and optimizing retail strategies.

Demonstration Store Selection

For the purpose of demonstrating the project and its outputs i have chosen the **Pharmacy Store** dataset from the available store options. The project code runs on various datasets but the results and analysis in this report are based on the transactions from the Pharmacy Store.

Screenshots

Screenshots of code execution, outputs, and dataset previews will be included here.

The CSV file

jupyter pharmacy.csv Last Checkpoint: 1 hour ago

File Edit View Settings Help

Delimiter: ,

	0	1	2	3	4
1	Razor	Toothpaste	Conditioner	Lotion	Shampoo
2	Towel	Mouthwash	Razor	Soap	Toothpaste
3	Towel	Deodorant	Mouthwash	Razor	
4	Toothpaste	Towel	Soap		
5	Conditioner	Soap	Toothbrush	Mouthwash	Towel
6	Lotion	Towel			
7	Mouthwash	Shampoo	Soap		
8	Soap	Toothpaste	Toothbrush		
9	Towel	Soap	Deodorant	Mouthwash	
10	Soap	Conditioner	Mouthwash	Deodorant	Toothbrush
11	Soap	Shampoo	Toothpaste		
12	Shampoo	Soap			
13	Towel	Toothpaste	Deodorant		
14	Soap	Toothpaste	Lotion	Conditioner	
15	Towel	Lotion	Soap	Shampoo	
16	Mouthwash	Lotion	Deodorant	Conditioner	
17	Soap	Conditioner	Toothbrush		
18	Toothbrush	Towel	Shampoo		
19	Toothbrush	Mouthwash	Soap	Towel	Conditioner
20	Toothpaste	Conditioner	Toothbrush	Razor	

1. **Store Selection and Input Prompt**

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

# storing paths to csv files for each of the stores
store_files = {
    1: ('Grocery Store', 'grocery_store.csv'),
    2: ('Pharmacy', 'pharmacy.csv'),
    3: ('Stationery Store', 'stationery_store.csv'),
    4: ('Clothing Store', 'clothing_store.csv'),
    5: ('Electronics Store', 'electronics_store.csv')
}

# store selection
print("Welcome to Apriori 2.0!")
print("User, please select your store:")
for key, value in store_files.items():
    print(f"{key}: {value[0]}")

store_choice = int(input("Enter the store number: "))
store_name, file_name = store_files[store_choice]
print(f"You have selected dataset located in {file_name}.")

# fetching the csv file for the selected store
df_transactions = pd.read_csv(file_name)
transactions = df_transactions.values.tolist()

# removing ("nan" - not a number) values from transactions
transactions = [[item for item in transaction if str(item) != 'nan'] for transaction in transactions]
```

2. Brute Force Frequent Itemsets

```
# results for Brute Force
print("\n---> Brute Force frequent Itemsets <---")
if frequent_itemsets_brute:
    for itemset, support in frequent_itemsets_brute:
        print(f"Itemset: {itemset}, Support: {support:.2f}")
else:
    print("No frequent itemsets found using Brute Force for the given support.")
print(f"Brute Force Execution Time: {end_brute_force - start_brute_force:.4f} seconds")
```

3. Apriori Frequent Itemsets

```
# Apriori Algorithm
te = TransactionEncoder()
te_ary = te.fit(transactions).transform(transactions)
df = pd.DataFrame(te_ary, columns=te.columns_)

start_apriori = time.time()
frequent_itemsets_apriori = apriori(df, min_support=min_support, use_colnames=True)
end_apriori = time.time()

# checkingif Apriori found any frequent itemsets
if frequent_itemsets_apriori.empty:
    print("No frequent itemsets found using Apriori for the given support.")
else:
    # print Apriori frequent itemsets
    print("\n---> Apriori Frequent Itemsets <---")
    for i, row in frequent_itemsets_apriori.iterrows():
        print(f"Itemset: {list(row['itemsets'])}, Support: {row['support']:.2f}")
```

4. Apriori Association Rules

```
    # generating association rules for Apriori
    rules_apriori = association_rules(frequent_itemsets_apriori, metric="confidence", min_threshold=min_confidence)
    if rules_apriori.empty:
        print("No association rules found using Apriori for the given confidence.")
    else:
        print("\nfinal Association Rules:")
        for i, row in rules_apriori.iterrows():
            antecedent = list(row['antecedents'])
            consequent = list(row['consequents'])
            confidence = row['confidence'] * 100
            support = row['support'] * 100
            print(f"Rule {i+1}: {antecedent} -> {consequent}")
            print(f"Confidence: {confidence:.2f}%")
            print(f"Support: {support:.2f}%\n")
```

5. FP-Tree Frequent Patterns and Association Rules

```
# FP-tree Algorithm
start_fp = time.time()
patterns_fp = pyfpgrowth.find_frequent_patterns(transactions, int(min_support * len(transactions)))
end_fp = time.time()

# fp-tree Frequent Patterns
print("\n---> FP-tree Frequent Patterns <---")
for itemset, support in patterns_fp.items():
    print(f"Pattern: {itemset}, Support: {support / len(transactions):.2f}")

# handle case where no frequent patterns are found by fp-tree
if not patterns_fp:
    print("No frequent patterns found using FP-tree for the given support.")
else:
    rules_fp = pyfpgrowth.generate_association_rules(patterns_fp, min_confidence)
    if not rules_fp:
        print("No association rules found using FP-tree for the given confidence.")
    else:
        print("\nFinal fp-tree Association Rules:")
        for i, (antecedent, (consequent, confidence)) in enumerate(rules_fp.items()):
            print(f"Rule {i+1}: {antecedent} -> {consequent}")
            print(f"Confidence: {confidence * 100:.2f}%")
            print(f"Support: {patterns_fp[antecedent] / len(transactions) * 100:.2f}%\n")
```

6. Performance Comparison

```python
# comparing the performance of the three algorithms
print(f"\nPerformance Comparison:")
print(f"Brute Force: {end_brute_force - start_brute_force:.4f} seconds")
print(f"Apriori: {end_apriori - start_apriori:.4f} seconds")
print(f"FP-tree: {end_fp - start_fp:.4f} seconds")
```

Output

```
Welcome to Apriori 2.0!
User, please select your store:
1: Grocery Store
2: Pharmacy
3: Stationery Store
4: Clothing Store
5: Electronics Store
Enter the store number:  2
You have selected dataset located in pharmacy.csv.
Please enter the Minimum Support in (%) you want (value from 1 to 100):  30
please enter the Minimum Confidence in (%) you want (value from 1 to 100):  20
```

```
---> Apriori Frequent Itemsets <---
Itemset: ['Conditioner'], Support: 0.40
Itemset: ['Mouthwash'], Support: 0.40
Itemset: ['Shampoo'], Support: 0.30
Itemset: ['Soap'], Support: 0.65
Itemset: ['Toothbrush'], Support: 0.35
Itemset: ['Toothpaste'], Support: 0.40
Itemset: ['Towel'], Support: 0.50
Itemset: ['Mouthwash', 'Soap'], Support: 0.30
Itemset: ['Soap', 'Towel'], Support: 0.30
```

```
final Association Rules:
Rule 1: ['Mouthwash'] -> ['Soap']
Confidence: 75.00%
Support: 30.00%

Rule 2: ['Soap'] -> ['Mouthwash']
Confidence: 46.15%
Support: 30.00%

Rule 3: ['Soap'] -> ['Towel']
Confidence: 46.15%
Support: 30.00%

Rule 4: ['Towel'] -> ['Soap']
Confidence: 60.00%
Support: 30.00%
```

```
---> FP-tree Frequent Patterns <---
Pattern: ('Shampoo',), Support: 0.30
Pattern: ('Toothbrush',), Support: 0.35
Pattern: ('Toothpaste',), Support: 0.40
Pattern: ('Conditioner',), Support: 0.40
Pattern: ('Mouthwash',), Support: 0.40
Pattern: ('Mouthwash', 'Soap'), Support: 0.30
Pattern: ('Towel',), Support: 0.50
Pattern: ('Soap', 'Towel'), Support: 0.30
Pattern: ('Soap',), Support: 0.65
```

```
Final fp-tree Association Rules:
Rule 1: ('Mouthwash',) -> ('Soap',)
Confidence: 75.00%
Support: 40.00%

Rule 2: ('Soap',) -> ('Towel',)
Confidence: 46.15%
Support: 65.00%

Rule 3: ('Towel',) -> ('Soap',)
Confidence: 60.00%
Support: 50.00%
```

```
---> Brute Force frequent Itemsets <---
Itemset: ('Towel',), Support: 0.50
Itemset: ('Toothpaste',), Support: 0.40
Itemset: ('Toothbrush',), Support: 0.35
Itemset: ('Conditioner',), Support: 0.40
Itemset: ('Soap',), Support: 0.65
Itemset: ('Shampoo',), Support: 0.30
Itemset: ('Mouthwash',), Support: 0.40
Itemset: ('Towel', 'Soap'), Support: 0.30
Itemset: ('Soap', 'Mouthwash'), Support: 0.30
Brute Force Execution Time: 0.0034 seconds
```

```
Performance Comparison:
Brute Force: 0.0034 seconds
Apriori: 0.0030 seconds
FP-tree: 0.0003 seconds
```

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/vm675/DataMining_Midterm_Project