Name :- Vansh Vig
UCID :- vv455
Email :- vv455@njit.edu
Date :- 10/11/2024
Professor : - Yasser Abduallah
CS634 :- **Data Mining**

Midterm Project Report

Implementation and Code Usage

Apriori Algorithm Implementation in Retail Data Mining:

i. Abstract :-

In this study, I employ both the Apriori Algorithm and FP-Growth algorithm to detect relationships in retail transactions. I assessed the algorithm's efficiency and effectiveness by applying various data mining methods and strategies. In addition, I create a personalized framework to uncover important information from transaction data using design.

ii. Introduction :-

Within the field of data mining, it is crucial to identify connections and trends in extensive sets of data for effective decision-making. This project seeks to investigate and contrast different algorithms for discovering association rules and frequent patterns. By combining brute force tactics with Apriori and FP-Growth algorithms, we explore the domain of association clustering. Through the utilization of these algorithms, our goal is to extract important information from transactional data and assess how successful each method is in recognizing significant connections. By conducting this comparative analysis, we aim to illuminate the pros and cons of various methods, offering a thorough grasp of their relevance in practical situations. Our goal is to highlight the effectiveness and importance of data mining techniques in revealing hidden patterns that are crucial for making informed decisions, through presenting the outcomes of each algorithm.

Subsequent Procedures :

- Step 1: Cloning the repository or saving the folder in local storage and executing requirements.txt.
- Step 2: Selecting the dataset and itemsets from CSV files.
- Step 3: Processing the dataset
- Sep 4: Gathering user input for minimum support and confidence thresholds.
- Step 5: Sequentially producing candidate itemsets and refining frequent itemsets through the Brute Force approach, apriori and FPGrowth tree and comparing the results

iii. Foundational Concepts and Principles:

Discovery of Common Itemsets:

The essence of the association algorithms lies in uncovering common itemsets, denoting groups of items recurrently appearing together in transactions. These sets offer valuable insights into customer purchasing patterns and preferences.

Support and Confidence:

In the realm of data mining, pivotal metrics include support and confidence. Support quantifies the frequency of occurrence for an item or itemset, while confidence evaluates the probability of items being bought in tandem. These metrics serve as guiding principles for our analytical endeavours.

Association Rules :

Through the identification of robust association rules, I ascertain which items are frequently bought in conjunction. These rules play a crucial role in enhancing sales strategies, including personalised recommendations.

iV. Project Workflow :

Our project follows a systematic process involving several steps and the use of the Apriori Algorithm.

Data Loading and Preprocessing :

We offer a variety of retail datasets such as Nike, K-Mart, and Amazon books, among others. The user has the option to select any of them. The dataset has been altered so that each column represents an item and each row represents a transaction. If an item is present in a transaction, the value is True; otherwise, it is Null. In the library implementation, we have replaced Null values with ' False' in the dataset as algorithms do not support Null values.

Establishing Minimum Support and Confidence Thresholds :

User input is extremely significant in the field of data mining. We request the user's input on the minimum support and confidence thresholds, crucial for filtering out less important patterns.

Iterating Through Candidate Itemsets :

The iterative procedure of implementing the Brute Force Algorithm involves creating candidate itemsets that increase in size step by step. We begin with single items (itemset size K = 1), then progress to K = 2, K = 3, and so on. This step-by-step process uses a systematic "brute force" method to thoroughly produce all possible combinations of itemsets.

Support Count Computation :

When we identify a candidate itemset, we calculate its support by counting the transactions containing that itemset. Itemsets that meet the minimum support threshold are kept, while those that do not meet the requirement are ignored.

Confidence Computation:

We evaluate the certainty of association rules, indicating the strength of relationships between items . This process requires a detailed comparison of support values related to single items and itemsets.

Association Rule Formation:
We identify association rules that meet the requirements for minimum support and minimum confidence. These guidelines reveal crucial information about the common practice of buying items together.

V. Results and Evaluation :

The project's effectiveness and efficiency are evaluated using performance metrics that include support, confidence, and the resulting association rules. Moreover, we compare our custom Brute Force Apriori Algorithm with the Apriori and FP Growth libraries to assess its reliability.

Vi. Conclusion :

To wrap up, our project demonstrates the real-life application of data mining concepts, beliefs, and methods. We successfully implemented the Apriori Algorithm to extract important association rules from retail transactional datasets. The effectiveness of data mining in revealing important patterns for informed decision-making in the retail industry is highlighted by the iterative process using a systematic "brute force" approach, customized algorithm design, and adherence to user-defined parameters.

Vii. Screenshots (Code Part) :

(A) Dataset : This is the Amazon Books Dataset, as we can see the column names are the items and the transactions where the particular item is present we have ‘t’ in that field

```
print(df.head())
```

```
  A Beginner's Guide Java: The Complete Reference Java For Dummies  \
0                t                         NaN              t   
1                t                         NaN              t   
2                t                         NaN            NaN   
3                t                         NaN              t   
4              NaN                           t              t   

  Android Programming: The Big Nerd Ranch Head First Java 2nd Edition  \
0                                NaN                            NaN   
1                                NaN                              t   
2                                NaN                              t   
3                                NaN                            NaN   
4                                NaN                              t   

  Beginning Programming with Java Java 8 Pocket Guide  \
0                               t                 t   
1                             NaN               NaN   
2                             NaN               NaN   
3                             NaN               NaN   
4                             NaN               NaN   

  C++ Programming in Easy Steps Effective Java (2nd Edition)  \
0                             t                            t   
1                           NaN                          NaN   
2                           NaN                            t   
3                           NaN                            t   
4                           NaN                          NaN   

  HTML and CSS: Design and Build Websites  
0                                     NaN  
1                                     NaN  
2                                     NaN  
3                                       t  
4                                       t  
```

(B) <u>The Data_Mining_MidTerm_Project.ipynb file</u>

Data Mining Midterm Project - Vansh Vig

Importing Libraries

[1]:
```python
import numpy as np
import pandas as pd
import itertools
from itertools import combinations
from collections import defaultdict
from mlxtend.frequent_patterns import apriori, association_rules, fpgrowth
import time
```

Importing Datasets

[2]:
```python
def read_dataset(choice):
    if choice == 1:
        print("Reading Amazon books dataset")
        df = pd.read_csv('AmazonBooks.csv')
    elif choice == 2:
        print("Reading BestBuy dataset")
        df = pd.read_csv('BestBuy.csv')
    elif choice == 3:
        print("Reading Generic dataset")
        df = pd.read_csv('Generic.csv')
    elif choice == 4:
        print("Reading Grocery Store dataset")
        df = pd.read_csv('Grocery Store.csv')
    elif choice == 5:
        print("Reading K-mart dataset")
        df = pd.read_csv('K-Mart.csv')
    elif choice == 6:
        print("Reading Nike dataset")
        df = pd.read_csv('Nike.csv')
    else:
        print("not a valid input, please select between 1 to 6")
        return None
    return df
```

Getting the Minimum Support and Minimum Confidence Values from the user

[64]:
```python
min_support_value = int(input("Enter the Minimum Support (eg. 30): "))
min_support = (min_support_value)/100
```

[65]:
```python
confidence = int(input("Enter the Minimun Confidence eg. (20) : "))
min_confidence = (confidence)/100
```

[66]:
```python
print("Minimum Support Value : ",min_support)
print("Minimum Confidence Value : ", min_confidence)
```

```
Minimum Support Value :  0.3
Minimum Confidence Value :  0.3
```

Importing the Brute Force code from the Brute_Force_Code.py file

[67]:
```python
from Brute_Force_Code import BruteForceAssociation

start_bruteforce = time.time()
associations = BruteForceAssociation(df, min_support, min_confidence)

# Print the association rules
print("Number of rules: ", len(associations), "\n\n")
for rule in associations:
    print(f"{set(rule[0])} -> {set(rule[1])} <confidence: {rule[2]}>")

end_bruteforce = time.time()
```

```
level : 1 2 3

Number of rules:  8


{'Java For Dummies'} -> {'A Beginner's Guide'} <confidence: 1.0>
{'A Beginner's Guide'} -> {'Java For Dummies'} <confidence: 1.0>
{'Java: The Complete Reference'} -> {'Beginning Programming with Java'} <confidence: 1.0>
{'Beginning Programming with Java'} -> {'Java: The Complete Reference'} <confidence: 1.0>
{'Android Programming: The Big Nerd Ranch'} -> {'Java 8 Pocket Guide'} <confidence: 1.0>
{'Java 8 Pocket Guide'} -> {'Android Programming: The Big Nerd Ranch'} <confidence: 1.0>
{'Java 8 Pocket Guide'} -> {'Beginning Programming with Java'} <confidence: 1.0>
{'Beginning Programming with Java'} -> {'Java 8 Pocket Guide'} <confidence: 1.0>
```

Time taken for brute force code

[68]:
```
time_taken = (end_bruteforce - start_bruteforce)*1000
print("Time taken:", time_taken, "ms")
```

```
Time taken: 4.065990447998047 ms
```

Processing the dataset to fit in the python library

[69]:
```
data = df.replace('t', True)
data = data.fillna(False)
```

Using the apriori and association_rules function from the mlxtend library

[70]:
```
start_apriori = time.time()
# Generate frequent itemsets
frequent_itemsets = apriori(data, min_support=min_support, use_colnames=True)  # Use column names if applicable

# Generate association rules
rules = association_rules(frequent_itemsets, metric="confidence", min_threshold=min_confidence)
print("No. of rules :" , len(rules))

end_apriori = time.time()
```

```
No. of rules : 8
```

Time taken for Apriori

[71]:
```
time_taken = (end_apriori - start_apriori)*1000
print("Time taken for Apriori Algorithm:", time_taken, "ms")
```

```
Time taken for Apriori Algorithm: 13.19265365600586 ms
```

Using the fp-growth algorithm from the mlxtend library

[73]:
```
start_fpgrowth = time.time()
frequent_itemsets_fpgrowth = fpgrowth(data, min_support=min_support, use_colnames=True)  # Use column names if applicable

# Generate association rules
rules_fpgrowth = association_rules(frequent_itemsets_fpgrowth, metric="confidence", min_threshold=min_confidence)
print("No. of rules :", len(rules_fpgrowth))

end_fpgrowth = time.time()
```

```
No. of rules : 8
```

Time taken for FP Growth Algorithm

[74]:
```
time_taken = (end_fpgrowth - start_fpgrowth)*1000
print("Time taken for FP-growth Algorithm:", time_taken, "ms")
```

```
Time taken for FP-growth Algorithm: 6.573200225830078 ms
```

Result of FP Growth Algorithm

[75]:
```
print("Frequent itemsets:")
print(frequent_itemsets_fpgrowth.head())  # Display the first few items

print("\nAssociation rules:")
print(rules_fpgrowth[['antecedents', 'consequents', 'support', 'confidence']].head(10))  # Display the first few rules
```

```
Frequent itemsets:
    support                     itemsets
0  0.473684          (Java 8 Pocket Guide)
1  0.421053  (Beginning Programming with Java)
2  0.421053            (A Beginner's Guide)
3  0.368421              (Java For Dummies)
```

(C) The Brute_Force_Code.py (Working of each part is explained in the comments of the code)

```python
from collections import defaultdict
from itertools import combinations

def BruteForceAssociation(dataset, min_support, min_confidence):
    # Function to find frequent itemsets and association rules using a brute force approach

    # Extracting item list from dataset
    item_list = list(dataset.columns)

    # Mapping each item to an integer value
    item_count = dict()
    for i, item in enumerate(item_list):
        item_count[item] = i + 1

    # Converting dataset to a list of transactions
    transaction_list = list()
    for index, record in dataset.iterrows():
        current_transaction = set()
        for key in item_count:
            if record[key] == 't':
                current_transaction.add(item_count[key])
        transaction_list.append(current_transaction)

    # Function to calculate support of an itemset in transaction list
    def get_support(transaction_list, item_set):
        match_count = 0
        for transaction in transaction_list:
            if item_set.issubset(transaction):
                match_count += 1
        return float(match_count / len(transaction_list))

    # Function to generate candidate itemsets at each level
    def self_join_candidates(frequent_item_sets_per_level, level):
        current_level_candidates = list()
        last_level_items = frequent_item_sets_per_level[level - 1]
        if len(last_level_items) == 0:
            return current_level_candidates
        for i in range(len(last_level_items)):
            for j in range(i+1, len(last_level_items)):
                itemset_i = last_level_items[i][0]
```

```python
    # Function to generate single-drop subsets of an itemset
    def get_single_drop_subsets(item_set):
        single_drop_subsets = list()
        for item in item_set:
            temp = item_set.copy()
            temp.remove(item)
            single_drop_subsets.append(temp)
        return single_drop_subsets

    # Function to check if an itemset is valid based on its subsets
    def is_valid_set(item_set, prev_level_sets):
        single_drop_subsets = get_single_drop_subsets(item_set)
        for single_drop_set in single_drop_subsets:
            if single_drop_set not in prev_level_sets:
                return False
        return True

    # Function to prune candidate itemsets
    def prune_candidates(frequent_item_sets_per_level, level, candidate_set):
        post_pruning_set = list()
        if len(candidate_set) == 0:
            return post_pruning_set
        prev_level_sets = list()
        for item_set, _ in frequent_item_sets_per_level[level - 1]:
            prev_level_sets.append(item_set)
        for item_set in candidate_set:
            if is_valid_set(item_set, prev_level_sets):
                post_pruning_set.append(item_set)
        return post_pruning_set

    # Function to generate frequent itemsets
    def get_frequent_itemsets(min_support):
        frequent_item_sets_per_level = defaultdict(list)
        print("level : 1", end=" ")
        for item in range(1, len(item_list) + 1):
            support = get_support(transaction_list, {item})
            if support >= min_support:
                frequent_item_sets_per_level[1].append(({item}, support))
        for level in range(2, len(item_list) + 1):
            print(level, end=" ")
```

```python
                frequent_item_sets_per_level[1].append(({item}, support))
        for level in range(2, len(item_list) + 1):
            print(level, end=" ")
            current_level_candidates = self_join_candidates(frequent_item_sets_per_level, level)
            post_pruning_candidates = prune_candidates(frequent_item_sets_per_level, level, current_level_candidates)
            if len(post_pruning_candidates) == 0:
                break
            for item_set in post_pruning_candidates:
                support = get_support(transaction_list, item_set)
                if support >= min_support:
                    frequent_item_sets_per_level[level].append((item_set, support))
        return frequent_item_sets_per_level

    # Generating frequent itemsets
    frequent_item_sets_per_level = get_frequent_itemsets(min_support)
    print("\n")

    # Function to generate association rules from frequent itemsets
    def get_association_rules(min_confidence, support_dict):
        rules = []
        for item, support in support_dict.items():
            if len(item) > 1:
                subsets = subset_finder(item, len(item))
                for subset in subsets:
                    complement = item.difference(subset)
                    if complement:
                        subset = frozenset(subset)
                        rule = (subset, frozenset(complement), support_dict[frozenset(item)] / support)
                        if rule[2] >= min_confidence:
                            rules.append(rule)
        return rules

    # Generating association rules
    return get_association_rules(min_confidence, item_support_dict)
```

(D) Output Results for the K-Mart Dataset (Brute Force) for Min Support = 30 and Min Confidence = 30

```
level : 1 2 3 4

Number of rules:  32


{'Quilts'} -> {'Bedspreads'} <confidence: 1.0>
{'Bedspreads'} -> {'Quilts'} <confidence: 1.0>
{'Quilts'} -> {'Decorative Pillows'} <confidence: 1.0>
{'Decorative Pillows'} -> {'Quilts'} <confidence: 1.0>
{'Quilts'} -> {'Shams'} <confidence: 1.0>
{'Shams'} -> {'Quilts'} <confidence: 1.0>
{'Bedspreads'} -> {'Decorative Pillows'} <confidence: 1.0>
{'Decorative Pillows'} -> {'Bedspreads'} <confidence: 1.0>
{'Bedding Collections'} -> {'Bedspreads'} <confidence: 1.0>
{'Bedspreads'} -> {'Bedding Collections'} <confidence: 1.0>
{'Bedspreads'} -> {' Kids Bedding'} <confidence: 1.0>
{' Kids Bedding'} -> {'Bedspreads'} <confidence: 1.0>
{'Embroidered Bedspread'} -> {'Bedspreads'} <confidence: 1.0>
{'Bedspreads'} -> {'Embroidered Bedspread'} <confidence: 1.0>
{'Bed Skirts'} -> {'Decorative Pillows'} <confidence: 1.0>
{'Decorative Pillows'} -> {'Bed Skirts'} <confidence: 1.0>
{'Decorative Pillows'} -> {'Shams'} <confidence: 1.0>
{'Shams'} -> {'Decorative Pillows'} <confidence: 1.0>
{'Bedding Collections'} -> {'Decorative Pillows'} <confidence: 1.0>
{'Decorative Pillows'} -> {'Bedding Collections'} <confidence: 1.0>
{'Decorative Pillows'} -> {' Kids Bedding'} <confidence: 1.0>
{' Kids Bedding'} -> {'Decorative Pillows'} <confidence: 1.0>
{'Bedding Collections'} -> {'Bed Skirts'} <confidence: 1.0>
{'Bed Skirts'} -> {'Bedding Collections'} <confidence: 1.0>
{'Bedding Collections'} -> {' Kids Bedding'} <confidence: 1.0>
{' Kids Bedding'} -> {'Bedding Collections'} <confidence: 1.0>
{'Quilts'} -> {'Decorative Pillows', 'Shams'} <confidence: 1.0>
{'Decorative Pillows'} -> {'Quilts', 'Shams'} <confidence: 1.0>
{'Shams'} -> {'Quilts', 'Decorative Pillows'} <confidence: 1.0>
{'Quilts', 'Decorative Pillows'} -> {'Shams'} <confidence: 1.0>
{'Quilts', 'Shams'} -> {'Decorative Pillows'} <confidence: 1.0>
{'Decorative Pillows', 'Shams'} -> {'Quilts'} <confidence: 1.0>
```

(E) Output results for K-Mart Dataset (Apriori) for Min Support = 30 and Min Confidence = 30

```
No. of rules : 32
```

Time taken for Apriori

```python
time_taken = (end_apriori - start_apriori)*1000
print("Time taken for Apriori Algorithm:", time_taken, "ms")
```

```
Time taken for Apriori Algorithm: 13.561248779296875 ms
```

Results of apriori algorithm

```python
print("Frequent itemsets:")
print(frequent_itemsets.head())  # Display the first few items

print("\nAssociation rules:")
print(rules[['antecedents', 'consequents', 'support', 'confidence']].head(10))  # Display the first few rules
```

```
Frequent itemsets:
    support             itemsets
0  0.473684             (Quilts)
1  0.684211         (Bedspreads)
2  0.684211  (Decorative Pillows)
3  0.421053         (Bed Skirts)
4  0.473684              (Shams)

Association rules:
            antecedents            consequents   support  confidence
0              (Quilts)           (Bedspreads)  0.315789    0.666667
1          (Bedspreads)               (Quilts)  0.315789    0.461538
2              (Quilts)   (Decorative Pillows)  0.368421    0.777778
3  (Decorative Pillows)               (Quilts)  0.368421    0.538462
4              (Quilts)                (Shams)  0.368421    0.777778
5               (Shams)               (Quilts)  0.368421    0.777778
6          (Bedspreads)   (Decorative Pillows)  0.473684    0.692308
7  (Decorative Pillows)           (Bedspreads)  0.473684    0.692308
8  (Bedding Collections)          (Bedspreads)  0.368421    0.636364
9          (Bedspreads)  (Bedding Collections)  0.368421    0.538462
```

(F) Output results for K-Mart Dataset (FP Growth) for Min Support = 30 and Min Confidence = 30

```
No. of rules : 32
```

Time taken for FP Growth Algorithm

```python
[14]: time_taken = (end_fpgrowth - start_fpgrowth)*1000
      print("Time taken for FP-growth Algorithm:", time_taken, "ms")
```

```
Time taken for FP-growth Algorithm: 6.078481674194336 ms
```

Result of FP Growth Algorithm

```python
[15]: print("Frequent itemsets:")
      print(frequent_itemsets_fpgrowth.head())  # Display the first few items

      print("\nAssociation rules:")
      print(rules_fpgrowth[['antecedents', 'consequents', 'support', 'confidence']].head(10))  # Display the first few rules
```

```
Frequent itemsets:
    support              itemsets
0  0.684211   (Decorative Pillows)
1  0.578947  (Bedding Collections)
2  0.473684         ( Kids Bedding)
3  0.473684              (Quilts)
4  0.421053          (Bed Skirts)

Association rules:
             antecedents            consequents   support  confidence
0  (Bedding Collections)   (Decorative Pillows)  0.421053    0.727273
1   (Decorative Pillows)  (Bedding Collections)  0.421053    0.615385
2  (Bedding Collections)           (Bedspreads)  0.368421    0.636364
3           (Bedspreads)  (Bedding Collections)  0.368421    0.538462
4   (Decorative Pillows)         ( Kids Bedding)  0.421053    0.615385
5        ( Kids Bedding)   (Decorative Pillows)  0.421053    0.888889
6  (Bedding Collections)         ( Kids Bedding)  0.315789    0.545455
7        ( Kids Bedding)  (Bedding Collections)  0.315789    0.666667
8           (Bedspreads)         ( Kids Bedding)  0.315789    0.461538
9        ( Kids Bedding)           (Bedspreads)  0.315789    0.666667
```

GITHUB LINK:
https://github.com/vanshvigggg/Data-mining-mid-term-project-