NAME: Vishal Fenn

NJIT UCID: vkf

Email Address: vkf@njit.edu

<10/09/2024>

Professor: Yasser Abduallah

CS 634 <101> Data Mining

# Midterm Project Report

## Concepts:

- **Support** is calculated by taking the amount of times that an itemset appears in the transactions divided the total number of transactions. It is about the popularity of the item-set in relation to the transactions.
- **Confidence** is calculated by the amount of times that an itemset appears in the transactions divided by the support of one or more items from the itemset. It shows how likely two or more items are purchased together in a transaction.
- **Apriori** is an association rule mining algorithm that identifies frequent individual items in transaction and then extends the identified items to larger itemsets. The Brute Force algorithm in this project is based on the Apriori algorithm.
- **FP Growth** is a tree structure association rule mining algorithm that is more efficient than Apriori. It is implemented by reordering the itemsets by count and then it find the frequent patterns starting from the bottom nodes and traversing up to the root. The algorithm finds all combinations of frequent patterns to satisfy the minimum support for a conditional. This algorithm can be useful when it comes to dealing with larger databases and transactions.
- **Association Rule** mining is about finding frequent patterns or strong correlations among a set of items in a large database. The goal is to find all the rules from the given transactions in order to increase customer satisfaction so that it can help increase profit for businesses. In order to qualify as rule, the itemsets have to pass the threshold of minimum support and minimum confidence. T

## Tutorial

1. Python 3.12.6 is the version used for this program. Make sure to have this or the latest version of python 3 installed.
2. In order to run the program in python, clone this repository to your home directory, by opening a `cmd` terminal and then clone the repository to your location of choice, recommend putting it somewhere in the home directory.
   - `git clone https://github.com/VkfNJIT/DataMiningMidterm.git`
3. Change directory to the path of the cloned repository using `cd` `path/to/project` on the command line
4. List all the files and folders in the your path using the `ls` command.
5. Then you want make sure you have the dependencies installed in order to run the algorithm successfully, so enter `pip install -r requirements.txt`
6. Once that has been successfully completed, on your terminal type:

   `python MidtermAlgorithm.py`

7. You should have a prompt appear that ask for you enter the number of the itemset that you want the algorithm to run.
    - It must be an int value from 1 to 5,
    - It will then ask for you to enter a minimum support value from (1, 100)
    - It will ask for you to enter a minimum confidence value from (1, 100)

**Work-flow**

- In this code above, I am importing all the python libraries that are used in this program.
- I first find the frequent support for itemsets where k = 1.
- After that has been complete, I use the Itertools is a python library and it will be using the combinations function to get all of the combinations of itemsets when k > 1 to find the support and confidence values.
- I use dictionaries to store the values.
- I find frequent itemsets based on minimum support and minimum confidence values from the user.
- I then generate association rules for the frequent itemsets.
- I then utilize mlxtend to implement the apriori and fp growth libraries
- I also utilize mlxtend to implement association rules library for both apriori and fp growth

**Images**

| | Item # | Item Name |
|---|---|---|
| **0** | 1 | Hammer |
| **1** | 2 | Nails |
| **2** | 3 | Paint |
| **3** | 4 | Screwdriver |
| **4** | 5 | Pliers |
| **5** | 6 | Light Bulbs |
| **6** | 7 | Extension Cord |
| **7** | 8 | Garden Hose |
| **8** | 9 | Rake |
| **9** | 10 | Caulk |

| | Transaction ID | Transaction |
|---|---|---|
| **0** | Trans1 | Hammer, Nails |
| **1** | Trans2 | Paint, Screwdriver |
| **2** | Trans3 | Garden Hose, Rake |
| **3** | Trans4 | Light Bulbs, Extension Cord, Nails |
| **4** | Trans5 | Hammer, Pliers, Nails |
| **5** | Trans6 | Paint, Screwdriver, Caulk |
| **6** | Trans7 | Rake, Garden Hose, Pliers, Light Bulbs |
| **7** | Trans8 | Hammer, Nails, Caulk |
| **8** | Trans9 | Extension Cord, Light Bulbs |
| **9** | Trans10 | Paint, Screwdriver, Hammer, Pliers, Nails |
| **10** | Trans11 | Rake, Caulk |
| **11** | Trans12 | Garden Hose, Pliers, Light Bulbs |
| **12** | Trans13 | Nails, Paint, Caulk |
| **13** | Trans14 | Screwdriver, Extension Cord, Rake |
| **14** | Trans15 | Hammer, Nails, Pliers, Light Bulbs |
| **15** | Trans16 | Garden Hose, Paint, Rake |
| **16** | Trans17 | Hammer, Caulk |
| **17** | Trans18 | Nails, Extension Cord, Pliers |
| **18** | Trans19 | Rake, Light Bulbs, Screwdriver, Caulk |
| **19** | Trans20 | Hammer, Paint, Pliers, Nails, Extension Cord |

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

```python
selected_stores = {1: "amazon", 2: "best_buy", 3: "k-mart", 4: "nike", 5: "ace_hardware"}
try:
    selected_id = int(input(
    "Enter the store number for the dataset that you want:\n1. Amazon\n2. Best Buy\n3. K-mart\n4. Nike\n5. Ace Hardware\n"))
    if selected_id not in selected_stores.keys():
        print("invalid number, There are only 5 choices!Try again next time")
        sys.exit()
except ValueError:
    print("Invalid input! There are only 5 choices, please enter a valid number(1 to 5) next time")
    sys.exit()
item_names = pd.read_csv(f"{os.getcwd()}/Itemsets/{selected_stores[selected_id]}_items.csv")
transactions = pd.read_csv(f"{os.getcwd()}/Itemsets/{selected_stores[selected_id]}_transactions.csv")
print(f"You have selected the {selected_stores[selected_id]} dataset")

# Enter the minimum support and the minimum confidence
min_support = float(input("Please enter the minimum support percent that you want (1 to 100):\n"))
min_support /= 100
min_confidence = float(input("Please enter the minimum confidence percent that you want (1 to 100):\n"))
min_confidence /= 100
```

```python
def item_k_support_possibilities(item_names, k):

    item_k_arrange = combinations(item_names, k)
    possibilities_of_k_items = [item for item in item_k_arrange]
    return possibilities_of_k_items

def count_itemsets_for_k(current_itemset, transactions, k):
    item_k_filter = [name for name in current_itemset.keys()]
    item_k_frequent_names = item_k_support_possibilities(item_k_filter, k)
    itemset_k = {}
    for item in item_k_frequent_names:
        count_occ = sum(1 for transact in transactions if set(item).issubset(transact))
        itemset_k[tuple(item)] = float(count_occ) / len(transactions)
    return itemset_k
```

```python
def collect_frequent_itemset(unfilter_dict_k, min_support):
    filtered_dict = {}
    for key, val in unfilter_dict_k.items():
        if val >= min_support:
            itemsetkey = f'{key}'
            filtered_dict[key] = val
    return filtered_dict
```

```python
def get_itemsets_with_confidence(total_itemset_frequent, min_confidence):
    itemset_confidence = {}
    itemset_copy = total_itemset_frequent.copy()
    for key, val in total_itemset_frequent.items():
        if isinstance(key, tuple):
            if len(key) == 2:
                first = key[0]
                second = key[-1]
                confidence_val = val / total_itemset_frequent[first]
                if confidence_val >= min_confidence:
                    itemset_confidence[(first, second)] = confidence_val
                first_reverse = key[-1]
                second_reverse = key[0]
                confidence_val = val / itemset_copy[first_reverse]
                if confidence_val >= min_confidence:
                    itemset_confidence[(first_reverse, second_reverse)] = confidence_val
                    itemset_copy[(first_reverse, second_reverse)] = val

            elif len(key) > 2:
                for i in range(1, len(key)+1):
                    for first in combinations(list(key), i):
                        second = tuple(set(key).difference(set(first))) # This will get what comes after ->
                        if len(second) > 0:
                            first = tuple(sorted(first))
                            if first in itemset_copy:

                                confidence_val = float(val)/itemset_copy[first]
                                if confidence_val >= min_confidence:
                                    itemset_confidence[(first, second)] = confidence_val
                                    itemset_copy[(first, second)] = val
                            else:
                                if len(first) == 1:
                                    confidence_val = val / itemset_copy[first[0]]
                                    if confidence_val >= min_confidence:
                                        itemset_confidence[(first, second)] = confidence_val
                                        itemset_copy[(first, second)] = val
                                else:
                                    for item in total_itemset_frequent.keys():
                                        if len(set(item).difference(set(first))) == 0:
                                            confidence_val = val / itemset_copy[item]
                                            if confidence_val >= min_confidence:
                                                itemset_confidence[(first, second)] = confidence_val
                                                itemset_copy[(first, second)] = val

    return itemset_confidence, itemset_copy
```

```python
start_point = time.time()
itemset_k1 = item_names.set_index("Item Name").to_dict()["Item #"]

# This technique Only for the itemsets where k = 1
# Split the string by comma to seperate each string in a row
item_k1_names = [name for name in item_names["Item Name"]]

item_k1_count = transactions['Transaction'].str.split(", ").explode().value_cc

item_k1 = item_k1_count.to_dict()

# Get the support value for each itemset-1
for k, _ in itemset_k1.items():
    if k not in item_k1:
        itemset_k1[k] = float(0)
    else:
        itemset_k1[k] = float(item_k1[k]) / len(transactions["Transaction"])
itemset_frequent_k1 = collect_frequent_itemset(itemset_k1, min_support)
item_k = transactions['Transaction'].str.split(", ").to_list()
itemset_k = {}
itemset_frequent_k = itemset_frequent_k1
k_val = 2
updated_itemset = itemset_frequent_k1
while len(itemset_frequent_k) >= k_val:
    itemset_k = count_itemsets_for_k(itemset_frequent_k1, item_k, k_val)
    itemset_frequent_k = collect_frequent_itemset(itemset_k, min_support)
    updated_itemset.update(itemset_frequent_k)
    k_val += 1
```

```
for key_s, val_s in updated_itemset.items():
    print(f"Itemset: {key_s}, Support: {val_s}\n")
item_conf, item_supp = get_itemsets_with_confidence(updated_itemset, min_confidence)
print()
rule_ci = 1
for key_c,val_c in item_conf.items():
    if len(key_c) == 2 and val_c > 0:
        print(
        f"Rule {rule_ci}:{set(key_c[0:1])} -> {set(key_c[1:])}\nConfidence: {val_c*100:.2f}%\nSupport: {item_supp[key_c]*100:.2f}%")
        rule_ci += 1
        print()
    else:
        for i in range(len(key_c)-1):
            if len(val_c) > 0:
                print(
        f"Rule {rule_ci}:{set(key_c[0:i+1])} -> {set(key_c[i+1:])}\nConfidence: {val_c*100:.2f}%\nSupport: {item_supp[key_c]*100:.2f}%")
                rule_ci += 1
                print()
end_point = time.time()
total_time = end_point - start_point
print("Time Taken to Execute Brute Force: ", total_time)
```

```
te = TransactionEncoder()
te_ary = te.fit(item_k).transform(item_k)
dataframe = pd.DataFrame(te_ary, columns=te.columns_)
apriori_start_point = time.time()
checking_apriori = apriori(dataframe, min_support=min_support, use_colnames=True)
print()
print("Apriori Library")
print(checking_apriori)
if len(checking_apriori.index) > 0:
    ar_ap = association_rules(checking_apriori, metric='confidence', min_threshold=min_confidence)
    print()
    print("Apriori Association Rules Library")
    ar_ap = ar_ap[['antecedents', 'consequents', 'support', 'confidence']]
    print(ar_ap)
    print()
apriori_end_point = time.time()
total_apriori_time = apriori_end_point - apriori_start_point
print("Time taken to execute Apriori: ", total_apriori_time)
print()
fp_growth_start_point = time.time()
checking_fpgrowth = fpgrowth(dataframe, min_support=min_support, use_colnames=True)
print("FP Growth Library")
print(checking_fpgrowth)
if len(checking_fpgrowth.index) > 0:
    ar_fp = association_rules(checking_fpgrowth, metric='confidence', min_threshold=min_confidence)
    print()
    print("FP Growth Association Rules Library")
    ar_fp = ar_fp[['antecedents', 'consequents', 'support', 'confidence']]
    print(ar_fp)
    print()
fp_growth_end_point = time.time()
total_fp_time = fp_growth_end_point - fp_growth_start_point
print("Time taken to execute FP Growth: ", total_fp_time)
print()
```

## Results

## Amazon Brute Force

```
Enter the store number for the dataset that you want:
1. Amazon
2. Best Buy
3. K-mart
4. Nike
5. Ace Hardware
 1
You have selected the amazon dataset
Please enter the minimum support percent that you want (1 to 100):
 50
Please enter the minimum confidence percent that you want (1 to 100):
 80
Itemset: A Beginner’s Guide, Support: 0.55

Itemset: Java: The Complete Reference, Support: 0.5

Itemset: Java For Dummies, Support: 0.65

Itemset: Android Programming: The Big Nerd Ranch, Support: 0.65

Itemset: ('Java: The Complete Reference', 'Java For Dummies'), Support: 0.5


Rule 1:{'Java: The Complete Reference'} -> {'Java For Dummies'}
Confidence: 100.00%
Support: 50.00%
```

**Amazon Verified Libraries**

```
Apriori Library
   support                                          itemsets
0     0.55                               (A Beginner’s Guide)
1     0.65          (Android Programming: The Big Nerd Ranch)
2     0.65                                 (Java For Dummies)
3     0.50                     (Java: The Complete Reference)
4     0.50   (Java For Dummies, Java: The Complete Reference)

FP Tree Library
   support                                          itemsets
0     0.65                                 (Java For Dummies)
1     0.65          (Android Programming: The Big Nerd Ranch)
2     0.55                               (A Beginner’s Guide)
3     0.50                     (Java: The Complete Reference)
4     0.50   (Java For Dummies, Java: The Complete Reference)

Association Rules Library
                     antecedents          consequents  support  confidence
0  (Java: The Complete Reference)  (Java For Dummies)      0.5         1.0
```

**Best Buy Brute Force**

```
Enter the store number for the dataset that you want:
1. Amazon
2. Best Buy
3. K-mart
4. Nike
5. Ace Hardware
 2
You have selected the best_buy dataset
Please enter the minimum support percent that you want (1 to 100):
 60
Please enter the minimum confidence percent that you want (1 to 100):
 80
Itemset: Lab Top, Support: 0.6

Itemset: Flash Drive, Support: 0.65

Itemset: Lab Top Case, Support: 0.7

Itemset: Anti-Virus, Support: 0.7

Itemset: ('Lab Top Case', 'Anti-Virus'), Support: 0.6


Rule 1:{'Lab Top Case'} -> {'Anti-Virus'}
Confidence: 85.71%
Support: 60.00%

Rule 2:{'Anti-Virus'} -> {'Lab Top Case'}
Confidence: 85.71%
Support: 60.00%
```

**Best Buy Verified Libraries**

```
Apriori Library
   support                    itemsets
0     0.70                (Anti-Virus)
1     0.65               (Flash Drive)
2     0.60                   (Lab Top)
3     0.70              (Lab Top Case)
4     0.60  (Lab Top Case, Anti-Virus)

FP Tree Library
   support                    itemsets
0     0.70                (Anti-Virus)
1     0.65               (Flash Drive)
2     0.70              (Lab Top Case)
3     0.60                   (Lab Top)
4     0.60  (Lab Top Case, Anti-Virus)

Association Rules Library
      antecedents     consequents  support  confidence
0  (Lab Top Case)    (Anti-Virus)      0.6    0.857143
1    (Anti-Virus)  (Lab Top Case)      0.6    0.857143
```

**K-mart Brute Force**

```
Enter the store number for the dataset that you want:
1. Amazon
2. Best Buy
3. K-mart
4. Nike
5. Ace Hardware
 3
You have selected the k-mart dataset
Please enter the minimum support percent that you want (1 to 100):
 50
Please enter the minimum confidence percent that you want (1 to 100):
 70
Itemset: Decorative Pillows, Support: 0.5

Itemset: Bed Skirts, Support: 0.55

Itemset: Sheets, Support: 0.5

Itemset: Shams, Support: 0.55

Itemset: Kids Bedding, Support: 0.6

Itemset: ('Bed Skirts', 'Kids Bedding'), Support: 0.5

Itemset: ('Sheets', 'Kids Bedding'), Support: 0.5


Rule 1:{'Bed Skirts'} -> {'Kids Bedding'}
Confidence: 90.91%
Support: 50.00%

Rule 2:{'Kids Bedding'} -> {'Bed Skirts'}
Confidence: 83.33%
Support: 50.00%

Rule 3:{'Sheets'} -> {'Kids Bedding'}
Confidence: 100.00%
Support: 50.00%

Rule 4:{'Kids Bedding'} -> {'Sheets'}
Confidence: 83.33%
Support: 50.00%
```

## K-mart Verified Library

```
Apriori Library
   support                    itemsets
0     0.55                (Bed Skirts)
1     0.50        (Decorative Pillows)
2     0.60              (Kids Bedding)
3     0.55                     (Shams)
4     0.50                    (Sheets)
5     0.50  (Bed Skirts, Kids Bedding)
6     0.50      (Sheets, Kids Bedding)

FP Tree Library
   support                    itemsets
0     0.50        (Decorative Pillows)
1     0.60              (Kids Bedding)
2     0.55                (Bed Skirts)
3     0.55                     (Shams)
4     0.50                    (Sheets)
5     0.50  (Bed Skirts, Kids Bedding)
6     0.50      (Sheets, Kids Bedding)

Association Rules Library
      antecedents     consequents  support  confidence
0    (Bed Skirts)  (Kids Bedding)      0.5    0.909091
1  (Kids Bedding)    (Bed Skirts)      0.5    0.833333
2        (Sheets)  (Kids Bedding)      0.5    1.000000
3  (Kids Bedding)        (Sheets)      0.5    0.833333
```

**Nike Brute Force**

```
Enter the store number for the dataset that you want:
1. Amazon
2. Best Buy
3. K-mart
4. Nike
5. Ace Hardware
 4
You have selected the nike dataset
Please enter the minimum support percent that you want (1 to 100):
 55
Please enter the minimum confidence percent that you want (1 to 100):
 80
Itemset: Running Shoe, Support: 0.7

Itemset: Socks, Support: 0.65

Itemset: Swimming Shirt, Support: 0.55

Itemset: Rash Guard, Support: 0.6

Itemset: Sweatshirts, Support: 0.65

Itemset: ('Running Shoe', 'Socks'), Support: 0.55

Itemset: ('Running Shoe', 'Sweatshirts'), Support: 0.55

Itemset: ('Socks', 'Sweatshirts'), Support: 0.6


Rule 1:{'Socks'} -> {'Running Shoe'}
Confidence: 84.62%
Support: 55.00%

Rule 2:{'Sweatshirts'} -> {'Running Shoe'}
Confidence: 84.62%
Support: 55.00%

Rule 3:{'Socks'} -> {'Sweatshirts'}
Confidence: 92.31%
Support: 60.00%

Rule 4:{'Sweatshirts'} -> {'Socks'}
Confidence: 92.31%
Support: 60.00%
```

**Nike Verified Libraries**

```
Apriori Library
   support                       itemsets
0     0.60                   (Rash Guard)
1     0.70                (Running Shoe)
2     0.65                        (Socks)
3     0.65                 (Sweatshirts)
4     0.55               (Swimming Shirt)
5     0.55          (Running Shoe, Socks)
6     0.55  (Running Shoe, Sweatshirts)
7     0.60           (Socks, Sweatshirts)

FP Tree Library
   support                       itemsets
0     0.70                (Running Shoe)
1     0.65                        (Socks)
2     0.65                 (Sweatshirts)
3     0.60                   (Rash Guard)
4     0.55               (Swimming Shirt)
5     0.55          (Running Shoe, Socks)
6     0.60           (Socks, Sweatshirts)
7     0.55  (Running Shoe, Sweatshirts)

Association Rules Library
     antecedents     consequents  support  confidence
0        (Socks)  (Running Shoe)     0.55    0.846154
1  (Sweatshirts)  (Running Shoe)     0.55    0.846154
2        (Socks)   (Sweatshirts)     0.60    0.923077
3  (Sweatshirts)         (Socks)     0.60    0.923077
```

**Ace Hardware Brute Force**

```
Enter the store number for the dataset that you want:
1. Amazon
2. Best Buy
3. K-mart
4. Nike
5. Ace Hardware
 5
You have selected the ace_hardware dataset
Please enter the minimum support percent that you want (1 to 100):
 30
Please enter the minimum confidence percent that you want (1 to 100):
 60
Itemset: Hammer, Support: 0.35

Itemset: Nails, Support: 0.45

Itemset: Paint, Support: 0.3

Itemset: Pliers, Support: 0.35

Itemset: Light Bulbs, Support: 0.3

Itemset: Rake, Support: 0.3

Itemset: Caulk, Support: 0.3

Itemset: ('Hammer', 'Nails'), Support: 0.3


Rule 1:{'Hammer'} -> {'Nails'}
Confidence: 85.71%
Support: 30.00%

Rule 2:{'Nails'} -> {'Hammer'}
Confidence: 66.67%
Support: 30.00%
```

**Ace Hardware Verified Libraries**

```
Apriori Library
   support         itemsets
0     0.30          (Caulk)
1     0.35         (Hammer)
2     0.30    (Light Bulbs)
3     0.45          (Nails)
4     0.30          (Paint)
5     0.35         (Pliers)
6     0.30           (Rake)
7     0.30  (Nails, Hammer)

FP Tree Library
   support         itemsets
0     0.45          (Nails)
1     0.35         (Hammer)
2     0.30          (Paint)
3     0.30           (Rake)
4     0.30    (Light Bulbs)
5     0.35         (Pliers)
6     0.30          (Caulk)
7     0.30  (Nails, Hammer)

Association Rules Library
  antecedents consequents  support  confidence
0     (Nails)    (Hammer)      0.3    0.666667
1    (Hammer)     (Nails)      0.3    0.857143
```

Comparing Algorithm Performance

| Algorithm | Brute Force (seconds) | Apriori (seconds) | FP Growth (seconds) |
|---|---|---|---|
| Amazon (Support: 40% , Confidence: 80%) | 0.018967628479003906 | 0.05368471145629883 | 0.0339510440826416 |
| Nike (Support: 60%, Confidence: 80%) | 0.010674715042114258 | 0.042136430740356445 | 0.020827770233154297 |
| Amazon (Support: 20% , Confidence: 80%) | 0.04908585548400879 | 0.026778459548950195 | 0.021683692932128906 |
| Nike (Support: 40%, Confidence: 80%) | 0.26961851119995117 | 0.03657674789428711 | 0.032068490982055664 |

When comparing the performance of the three algorithms, the performance of the algorithms seem to be dependent on the dataset and its minimum support value. Brute Force was able to run more efficiently than both Apriori and FP Growth when the minimum support value was high enough for a given dataset (shown in the table above). However, when the threshold for minimum support was lowered for the same dataset (shown in the table above), Brute Force would take a longer time to terminate than Apriori and FP Growth. The FP algorithm was the most efficient of the three in this scenario. I assume that when it comes to generating association rules, the execution time changes for the algorithms.

https://github.com/VkfNJIT/DataMiningMidterm