Seniz Ozdemir

UCID: so9

Email: so9@njit.edu

10/13/2024

Professor: Yasser Abdullah

CS 634-101 Data Mining

Midterm Project Report

The Apriori Algorithm and Data Mining in Retail Databases

The Apriori Algorithm is a fundamental algorithm used in frequent itemset mining and association rule mining and is most visibly deployed in retail settings to improve user shopping experiences and increase profits. The algorithm involves discovering frequent itemsets in transactional databases and extending those itemsets into larger itemsets. Frequent itemsets are used to construct association rules. This project implements a version of the Apriori Algorithm (referred to as "brute force") to generate association rules based on example transactional databases with user defined support and confidence values.

Implementation Details

Association rule mining is a powerful tool for analyzing transactional databases and uncovering useful patterns from large datasets. The purpose of this project is to implement a version of the Apriori Algorithm for generating frequent itemsets to use for generating association rules.

The Apriori Algorithm begins by generating frequent 1-itemsets by calculating the support of single items as they appear in a transactional database and removing items that do not meet some minimum support value. Support measures how frequently an itemset appears in the transactions database. For the purposes of this project, the minimum support value is user-determined. For the purposes of this project, a "brute force" version of Apriori was implemented where all 1-itemsets were combined to create 2-itemsets by combining the items into every possible combination of sets, after which the support of the item sets was calculated. This process was continued for k-itemsets until no further frequent itemsets could be found. These itemsets were then used to generate association rules according to a confidence value determined by the user. Confidence is a measure of the likelihood of items being purchased together.

In this implementation, several arbitrarily generated data sets from five arbitrarily chosen stores are provided for a user to choose to perform association rule mining according to minimum support and confidence values provided by the user.

Commented code with detailed explanations of implementation decisions and functionality follows. To replicate these results, all used .csv files and the source code is provided. The source code requires the installation of the pandas and mlxtend Python libraries. The version of Python

used during this project is version 3.12.6. More detailed notes about using this program are located in the "Usage Notes" section of this report.

Code and Further Implementation Details

Below is the implementation for the ItemSet class.

```
#Represents a k-itemset
class ItemSet:
    def __init__(self, items):
        self.items = items #a set of strings (item names)
        self.support = 0
        self.supportPercent = 0
        self.allAntecedents = [] #a list of sets of item names for association rules
    def __str__(self):
        return f"{self.items} Support: {self.support}, {self.supportPercent}%"
```

An ItemSet object is created with one argument, a set of strings that consist of the names of items that appear in the dataset. The support attribute represents the support of an itemset as an integer, or may be more easily understood as the raw number of times the itemset appears in transactions within the transactional database. The supportPercent attribute represents the itemset's support value as a percentage. The allAntecedents value is used during association rule generation and contains a list of sets of items generated from the items attribute. Further explanation of this attribute can be found in the description of the association_rule function. A __str__ function is provided for debugging convenience.

Below is the implementation of the Association Rule class.

```
#Represents an association rule
class AssociationRule:
    def __init__(self, antecedent, consequent):
        self.antecedent = antecedent #left hand side of rule
        self.consequent = consequent #right hand side of rule
        self.confidence = 0 #a percentage
        self.ruleSupport = 0 #a percentage
    def __str__(self):
        return f"{self.antecedent} -> {self.consequent} Support: {self.ruleSupport}%, Confidence: {self.confidence}%"
```

An association rule is typically written in the form "X → Y", where X and Y are itemsets. For clarity, X, or the left hand side of the rule, will furthermore be referred to as the antecedent; similarly, Y, or the right hand side of the rule, will furthermore be referred to as the consequent. The antecedent and consequent attributes are sets of strings to represent these itemsets. The confidence and ruleSupport values are percentages, calculated later in the program.

Two helper functions follow: itemset_support and frequent_itemset. These are used to generate frequent itemsets and are separated for readability.

```
#Helper function to calculates support for itemsets
#Takes a set of ItemSet objects
def itemset_support(itemSet):
    #process given itemSet and remove sets with less than minimum support
    for items in itemSet:
        for trans in transactionsList:
            compare = items.items
            if(compare.issubset(trans)):
                items.support += 1
    for item in itemSet:
        item.supportPercent = item.support/total_transactions * 100

#Helper function to remove infrequent itemsets
def frequent_itemset(itemSet):
    ret = []
    for item in itemSet:
        if(item.supportPercent >= minSupport):
            ret.append(item)
    return ret
```

The itemset_support function takes a list of ItemSet objects and, for each itemset, and scans the database to find transactions in which the given itemset appears, incrementing its support value when it does. Finally, the support values for every ItemSet are calculated as a percentage. The frequent_itemset function checks a list of ItemSet objects and returns a list containing only those ItemSets in which the support value for the set satisfies the user defined minimum support.

These two helper functions are used for readability in the k_itemset function below.

```
#Returns frequent k-itemset by making every combination from the 1 itemset.
#Returns a list of ItemSet objects
def k_itemset(k, singleItems):
    temp = []
    for thing in singleItems:
        for name in thing.items:
            temp.append(name)
    combos = list(combinations(temp, k))
    ret = []
    for combo in combos:
        tempItem = ItemSet(set(combo))
        ret.append(tempItem)
    itemset_support(ret)
    frret = frequent_itemset(ret)
    return frret
```

k_itemset takes an integer value k and a list of ItemSets. K determines how many items will be in the generated itemset, and singleItems is a list of all 1-itemsets in the transactional data set used to generate k-itemsets. Only frequent k-itemsets are returned, and this list may be empty.

The association_rules function utilizes two helper functions, the first of which is antecedents, shown below.

```
#Helper function to generate all antecedents possible for an item set
def antecedents(itemSet):
    temp = []
    ret = []
    for string in itemSet.items:
        temp.append(string)
    for i in range(1, len(temp)):
        for combo in combinations(temp, i):
            ret.append(set(combo))
    itemSet.allAntecedents = ret
```

This function takes a frequent itemset and forms sets of items to use as antecedents for association rules. It stores these antecedents in the respective ItemSet object in order to provide quicker access to its support value (more on this when the association rules generation is explained).

The second helper function is findSupport, shown below.

```
#Helper function, takes a set of strings (item names)
def findSupport(find):
    for item in allGeneratedItemsets:
        if(item.items == find):
            return item.support
    return 0
```

findSupport performs a linear search on all possible itemsets given the data to return an itemset's support value. It is separated for readability purposes.

Finally, the association_rules function follows below. This function generates a list of association rules from a frequent ItemSet and returns them.

Support for an association rule is equivalent to the support of the itemset (or for the set of all items in the rule). Association rule confidence is calculated with the following equation:

$X \rightarrow Y = \frac{support(X,Y)}{support(X)}$ where X and Y are itemsets. For ease of reading, the equation is equivalent to $Antecdent \rightarrow Consequent = \frac{Support(itemset)}{support(antcedent)}$. The following function will be explained in the context of this equation.

```
#Generates association rules and calculates confidence
#Takes an ItemSet object
def association_Rules(generateFrom):
    ret = [] #list of association rule objects
    numerator = generateFrom.support
    denominator = 0
    antecedents(generateFrom)
    ants = generateFrom.allAntecedents
    for ant in ants:
        consequent = generateFrom.items.difference(ant)
        temp = AssociationRule(ant, consequent)
        denominator = findSupport(ant)
        temp.ruleSupport = generateFrom.supportPercent
        temp.confidence = numerator/denominator * 100
        ret.append(temp)
    return ret
```

The support of the itemset is available within the ItemSet object passed to the equation. This is the value “numerator” takes. Using the antecedents function, all possible antecedents of the itemset are generated. For each antecedent, an AssociationRule object is created, and the consequent is calculated by taking the set difference of the whole itemset with the antecedent. Then the confidence of the rule is calculated, and all association rules are returned in a list.

The following is the driver code of the main program and should be self-explanatory. Here is the code that gets user input, looping until valid inputs are given:

```
#Intro
print("Available stores are listed below.\n")
print("1. Barnes and Noble\n2. Citadel Paints\n3. GameStop\n4. Staples\n5. Warhammer\n")

itemsAvailable = ''
transactions = ''

#Prompt user for valid selection
while(True):
    selection = input("Enter a number to select one of these stores, or enter 'q' to quit:\n")
    if(selection == 'q'):
        exit()

    selection = int(selection)

    if(selection == 1):
        itemsAvailable = 'barnes_and_noble.csv'
        transactions = 'barnes_and_noble_transactions.csv'
        break
    elif(selection == 2):
        itemsAvailable = 'citadel_paints.csv'
        transactions = 'citadel_paints_transactions.csv'
        break
    elif(selection == 3):
        itemsAvailable = 'gamestop.csv'
        transactions = 'gamestop_transactions.csv'
        break
    elif(selection == 4):
        itemsAvailable = 'staples.csv'
        transactions = 'staples_transactions.csv'
        break
    elif(selection == 5):
        itemsAvailable = 'warhammer.csv'
        transactions = 'warhammer_transactions.csv'
        break
    else:
        print("Invalid selection")
```

This code prompts users for a store selection and sets the .csv datasets used further in the program.

```
#Prompt user for support and confidence
while(True):
    minSupport = float(input("Please enter a minimum support value from 1 to 100 (this value is interpreted as a percentage):\n"))
    if(minSupport < 1 or minSupport > 100):
        print("Please enter a valid input from 1 to 100")
        continue
    break

while(True):
    minConfidence = float(input("Please enter a minimum confidence value from 1 to 100 (this value is interpreted as a percentage):\n"))
    if(minConfidence < 1 or minConfidence > 100):
        print("Please enter a valid input from 1 to 100")
        continue
    break
```

The user is then prompted for valid support and confidence values.

Data preprocessing is pictured below.

```
bruteStartTime = time.time()
#User defined input files
read_items = pd.read_csv(itemsAvailable, usecols=[1]) #Only item names are relevant, expect item names to be unique
read_transactions = pd.read_csv(transactions)

total_transactions = len(read_transactions)

#preprocess transactions into a list of sets (for usage of subset methods)
transactionsList = []
for x in range(total_transactions):
    tempTransaction = read_transactions.at[x, 'Transaction']
    tempTransaction = tempTransaction.split(', ')
    transactionsList.append(set(tempTransaction))

#Initial 1-itemsets in a list
oneItemsets = []
for i in range(len(read_items)):
    itemsInSet = {read_items.at[i, 'Item Name']}
    tempSet = ItemSet(itemsInSet)
    oneItemsets.append(tempSet)

itemset_support(oneItemsets)

allGeneratedItemsets = []
tempSets = []
```

All items a store sells are read from the .csv files indicated by the user's selection. The format of these files is specified in the "Usage Notes" section of this report. A list of transactions are created wherein every transaction is represented as a set, due to the fact that the itemset_support function checks if an itemset is a subset of a transaction set to determine support. 1-itemsets are constructed by parsing the .csv file of the store that contains its items on sale, and this set is then processed to create a list of 1-itemsets of ItemSet objects using itemset_support.

```
#Generate frequent k-itemsets
for i in range(1, len(read_items)):
    tempSets = k_itemset(i, oneItemsets)
    if(len(tempSets) == 0):
        break
    else:
        allGeneratedItemsets += tempSets

#Generate association rules
assocRules = []
for item in allGeneratedItemsets:
    assocRules += association_Rules(item)

if(len(assocRules) == 0):
    print("No rules found. Try a lower minimum support value (25 or less is realistic for these data sets)
else:
    print("Association rules (X -> Y read 'X implies Y'):")

for rule in assocRules:
    if(rule.confidence >= minConfidence):
        print(rule)
bruteEndTime = time.time()

print('\nResults returned in ', (bruteEndTime - bruteStartTime),'seconds from brute force algorithm\n')
```

All frequent k-itemsets are found by running a loop to find 1-itemsets, then 2-itemsets, and so on until no frequent k-itemsets are returned, at which point the loop terminates. Using these frequent itemsets, association rules are generated as explained previously, and the rules that meet the user defined minimum confidence level are displayed.

This experiment was also performed with mlxtend implementations of Apriori and FPgrowth for comparison and result confirmation. The driver code for both is below and should be self-explanatory.

```
#apriori from mlxtend for comparison
#data preprocessing for mlxtend
print('mlxtend Implementations for comparison purposes:\n')
aprioriStart = time.time()
dataset = []
for x in range(total_transactions):
    tempTrans= read_transactions.at[x, 'Transaction']
    tempTrans = tempTrans.split(', ')
    dataset.append(list(tempTrans))

te = TransactionEncoder()
te_ary = te.fit(dataset).transform(dataset)
df = pd.DataFrame(te_ary, columns=te.columns_)

apriori_generated = apriori(df, min_support=(minSupport/100), use_colnames=True)

rules = association_rules(apriori_generated, metric="confidence", min_threshold=(minConfidence/100))
rules = rules[['antecedents', 'consequents', 'support', 'confidence']]
aprioriEnd = time.time()
print("Association rules: ", rules, "\n")
print('Results returned in ', (aprioriEnd - aprioriStart),'seconds from mlxtend apriori implementation\n')

#fpgrowth from mlxtend for comparison

fpStart = time.time()
fpgrowth_generated = fpgrowth(df, min_support=(minSupport/100), use_colnames=True)
rules = association_rules(fpgrowth_generated, metric="confidence", min_threshold=(minConfidence/100))
rules = rules[['antecedents', 'consequents', 'support', 'confidence']]
fpEnd = time.time()
print("Association rules: ", rules, "\n")
print('Results returned in ', (fpEnd - fpStart),'seconds from mlxtend fpgrowth implementation')
```

Accuracy and Performance Comparisons

Here are some results of running the program for every store's dataset. Note that the staples_transactions.csv file contains 100 transactions for the purposes of comparing runtimes between the different algorithms used.

Analyzing these results, we see that the brute force algorithm implemented produces the same association rules as the mlxtend Apriori and fpgrowth algorithms, confirming its veracity. Through these experiments we see that the brute force implementation takes the longest time to compute the rules, the mlxtend Apriori implementation is faster than the brute force algorithm, and the mlxtend fpgrowth algorithm is the fastest of all three algorithms. The speed of the brute force algorithm is likely due to the lack of the usage of the Apriori principle in generating frequent itemsets, with all possible k-itemsets being generated from the available items every time instead. The brute force algorithm was also not written to be perfectly optimized, so some code is likely less efficiently implemented than the implementations used from existing python libraries. This time complexity comparison is easiest to see when using dataset 4, which contains a large number of transactions.

```
midterm_repo/cs634_midterm_proj/ozdemir_seniz_midtermproj.py"
Available stores are listed below.

1. Barnes and Noble
2. Citadel Paints
3. GameStop
4. Staples
5. Warhammer

Enter a number to select one of these stores, or enter 'q' to quit:
1
Please enter a minimum support value from 1 to 100 (this value is interpreted as a percentage):
15
Please enter a minimum confidence value from 1 to 100 (this value is interpreted as a percentage):
60
Association rules (X -> Y read 'X implies Y'):
{'Death of a Ferryman'} -> {'Gideon the Ninth'} Support: 15.0%, Confidence: 60.0%
{'Gideon the Ninth'} -> {'The Traitor Baru Cormorant'} Support: 25.0%, Confidence: 62.5%
{'The Traitor Baru Cormorant'} -> {'Gideon the Ninth'} Support: 25.0%, Confidence: 71.42857142857143%
{'The Left Hand of Darkness'} -> {'Oryx and Crake'} Support: 15.0%, Confidence: 60.0%
{'The Left Hand of Darkness'} -> {'Lonesome Dove'} Support: 15.0%, Confidence: 60.0%

Results returned in  0.006392478942871094 seconds from brute force algorithm

mlxtend Implementations for comparison purposes:

Association rules:                    antecedents                 consequents  support  confidence
0         (Death of a Ferryman)           (Gideon the Ninth)     0.15    0.600000
1            (Gideon the Ninth)  (The Traitor Baru Cormorant)     0.25    0.625000
2  (The Traitor Baru Cormorant)           (Gideon the Ninth)     0.25    0.714286
3   (The Left Hand of Darkness)              (Lonesome Dove)     0.15    0.600000
4   (The Left Hand of Darkness)              (Oryx and Crake)     0.15    0.600000

Results returned in  0.002993345260620117 seconds from mlxtend apriori implementation

Association rules:                    antecedents                 consequents  support  confidence
0   (The Left Hand of Darkness)              (Oryx and Crake)     0.15    0.600000
1   (The Left Hand of Darkness)              (Lonesome Dove)     0.15    0.600000
2         (Death of a Ferryman)           (Gideon the Ninth)     0.15    0.600000
3            (Gideon the Ninth)  (The Traitor Baru Cormorant)     0.25    0.625000
4  (The Traitor Baru Cormorant)           (Gideon the Ninth)     0.25    0.714286

Results returned in  0.003990650177001953 seconds from mlxtend fpgrowth implementation
PS C:\Users\ozdem\Desktop\CS 634\midterm_repo\cs634_midterm_proj>
```

```
midterm_repo/cs634_midterm_proj/ozdemir_seniz_midtermproj.py"
Available stores are listed below.

1. Barnes and Noble
2. Citadel Paints
3. GameStop
4. Staples
5. Warhammer

Enter a number to select one of these stores, or enter 'q' to quit:
2
Please enter a minimum support value from 1 to 100 (this value is interpreted as a percentage):
25
Please enter a minimum confidence value from 1 to 100 (this value is interpreted as a percentage):
50
Association rules (X -> Y read 'X implies Y'):
{'Nuln Oil'} -> {'Layer Brush'} Support: 30.0%, Confidence: 54.54545454545454%
{'Layer Brush'} -> {'Nuln Oil'} Support: 30.0%, Confidence: 66.66666666666666%
{'Dry Brush'} -> {'Nuln Oil'} Support: 30.0%, Confidence: 66.66666666666666%
{'Nuln Oil'} -> {'Dry Brush'} Support: 30.0%, Confidence: 54.54545454545454%
{'Agrax Earthshade'} -> {'Nuln Oil'} Support: 25.0%, Confidence: 83.33333333333334%

Results returned in  0.006036520004272461 seconds from brute force algorithm

mlxtend Implementations for comparison purposes:

Association rules:           antecedents    consequents  support  confidence
0  (Agrax Earthshade)     (Nuln Oil)     0.25    0.833333
1         (Dry Brush)     (Nuln Oil)     0.30    0.666667
2          (Nuln Oil)    (Dry Brush)     0.30    0.545455
3          (Nuln Oil)  (Layer Brush)     0.30    0.545455
4       (Layer Brush)     (Nuln Oil)     0.30    0.666667

Results returned in  0.005974531173706055 seconds from mlxtend apriori implementation

Association rules:           antecedents    consequents  support  confidence
0         (Dry Brush)     (Nuln Oil)     0.30    0.666667
1          (Nuln Oil)    (Dry Brush)     0.30    0.545455
2          (Nuln Oil)  (Layer Brush)     0.30    0.545455
3       (Layer Brush)     (Nuln Oil)     0.30    0.666667
4  (Agrax Earthshade)     (Nuln Oil)     0.25    0.833333

Results returned in  0.001995563507080078 seconds from mlxtend fpgrowth implementation
PS C:\Users\ozdem\Desktop\CS 634\midterm_repo\cs634_midterm_proj>
```

```
midterm_repo/cs634_midterm_proj/ozdemir_seniz_midtermproj.py"
Available stores are listed below.

1. Barnes and Noble
2. Citadel Paints
3. GameStop
4. Staples
5. Warhammer

Enter a number to select one of these stores, or enter 'q' to quit:
3
Please enter a minimum support value from 1 to 100 (this value is interpreted as a percentage):
20
Please enter a minimum confidence value from 1 to 100 (this value is interpreted as a percentage):
55
Association rules (X -> Y read 'X implies Y'):
{'Dualsense Controller'} -> {'Weird West'} Support: 25.0%, Confidence: 55.55555555555556%
{'Weird West'} -> {'Dualsense Controller'} Support: 25.0%, Confidence: 62.5%
{'Elden Ring'} -> {'Dualsense Controller'} Support: 20.0%, Confidence: 66.66666666666666%

Results returned in  0.0058133602142333984 seconds from brute force algorithm

mlxtend Implementations for comparison purposes:

Association rules:               antecedents            consequents  support  confidence
0            (Elden Ring)  (Dualsense Controller)     0.20    0.666667
1            (Weird West)  (Dualsense Controller)     0.25    0.625000
2  (Dualsense Controller)            (Weird West)     0.25    0.555556

Results returned in  0.0029931068420410156 seconds from mlxtend apriori implementation

Association rules:               antecedents            consequents  support  confidence
0            (Weird West)  (Dualsense Controller)     0.25    0.625000
1  (Dualsense Controller)            (Weird West)     0.25    0.555556
2            (Elden Ring)  (Dualsense Controller)     0.20    0.666667

Results returned in  0.0029571056365966797 seconds from mlxtend fpgrowth implementation
PS C:\Users\ozdem\Desktop\CS 634\midterm_repo\cs634_midterm_proj>
```

```
midterm_repo/cs634_midterm_proj/ozdemir_seniz_midtermproj.py"
Available stores are listed below.

1. Barnes and Noble
2. Citadel Paints
3. GameStop
4. Staples
5. Warhammer

Enter a number to select one of these stores, or enter 'q' to quit:
4
Please enter a minimum support value from 1 to 100 (this value is interpreted as a percentage):
10
Please enter a minimum confidence value from 1 to 100 (this value is interpreted as a percentage):
45
Association rules (X -> Y read 'X implies Y'):
{'Sticky Notes'} -> {'Pilot G2 Pens'} Support: 17.0%, Confidence: 50.0%
{'Pilot G2 Pens'} -> {'Sticky Notes'} Support: 17.0%, Confidence: 50.0%
{'Spiral Notebook'} -> {'Sticky Notes'} Support: 14.000000000000002%, Confidence: 48.275862068965516%
{'Printer Paper'} -> {'Stapler'} Support: 17.0%, Confidence: 45.94594594594595%
{'Stapler'} -> {'Printer Paper'} Support: 17.0%, Confidence: 54.83870967741935%
{'Bluetooth Mouse'} -> {'Stapler'} Support: 14.000000000000002%, Confidence: 50.0%
{'Stapler'} -> {'Bluetooth Mouse'} Support: 14.000000000000002%, Confidence: 45.16129032258064%

Results returned in  0.018700838088989258 seconds from brute force algorithm

mlxtend Implementations for comparison purposes:

Association rules:           antecedents        consequents  support  confidence
0  (Bluetooth Mouse)          (Stapler)     0.14    0.500000
1          (Stapler)  (Bluetooth Mouse)     0.14    0.451613
2     (Sticky Notes)    (Pilot G2 Pens)     0.17    0.500000
3    (Pilot G2 Pens)     (Sticky Notes)     0.17    0.500000
4    (Printer Paper)          (Stapler)     0.17    0.459459
5          (Stapler)    (Printer Paper)     0.17    0.548387
6  (Spiral Notebook)     (Sticky Notes)     0.14    0.482759

Results returned in  0.003990650177001953 seconds from mlxtend apriori implementation

Association rules:           antecedents        consequents  support  confidence
0     (Sticky Notes)    (Pilot G2 Pens)     0.17    0.500000
1    (Pilot G2 Pens)     (Sticky Notes)     0.17    0.500000
2  (Spiral Notebook)     (Sticky Notes)     0.14    0.482759
3    (Printer Paper)          (Stapler)     0.17    0.459459
4          (Stapler)    (Printer Paper)     0.17    0.548387
5  (Bluetooth Mouse)          (Stapler)     0.14    0.500000
6          (Stapler)  (Bluetooth Mouse)     0.14    0.451613

Results returned in  0.0029931068420410156 seconds from mlxtend fpgrowth implementation
PS C:\Users\ozdem\Desktop\CS 634\midterm_repo\cs634_midterm_proj>
```

```
Available stores are listed below.

1. Barnes and Noble
2. Citadel Paints
3. GameStop
4. Staples
5. Warhammer

Enter a number to select one of these stores, or enter 'q' to quit:
5
Please enter a minimum support value from 1 to 100 (this value is interpreted as a percentage):
20
Please enter a minimum confidence value from 1 to 100 (this value is interpreted as a percentage):
50
Association rules (X -> Y read 'X implies Y'):
{'Skaven Paint Set'} -> {'Stormcast Eternals Paint Set'} Support: 25.0%, Confidence: 50.0%
{'Stormcast Eternals Paint Set'} -> {'Skaven Paint Set'} Support: 25.0%, Confidence: 55.55555555555556%
{'Skaventide Box Set'} -> {'Skaven Paint Set'} Support: 20.0%, Confidence: 66.66666666666666%
{'Skaven Paint Set'} -> {'Skaven Dice Set'} Support: 25.0%, Confidence: 50.0%
{'Skaven Dice Set'} -> {'Skaven Paint Set'} Support: 25.0%, Confidence: 62.5%

Results returned in  0.018055438995361328 seconds from brute force algorithm

mlxtend Implementations for comparison purposes:

Association rules:                      antecedents                      consequents  support  confidence
0              (Skaven Paint Set)              (Skaven Dice Set)     0.25    0.500000
1               (Skaven Dice Set)             (Skaven Paint Set)     0.25    0.625000
2            (Skaventide Box Set)             (Skaven Paint Set)     0.20    0.666667
3              (Skaven Paint Set)  (Stormcast Eternals Paint Set)     0.25    0.500000
4  (Stormcast Eternals Paint Set)             (Skaven Paint Set)     0.25    0.555556

Results returned in  0.0049896240234375 seconds from mlxtend apriori implementation

Association rules:                      antecedents                      consequents  support  confidence
0              (Skaven Paint Set)  (Stormcast Eternals Paint Set)     0.25    0.500000
1  (Stormcast Eternals Paint Set)             (Skaven Paint Set)     0.25    0.555556
2              (Skaven Paint Set)              (Skaven Dice Set)     0.25    0.500000
3               (Skaven Dice Set)             (Skaven Paint Set)     0.25    0.625000
4            (Skaventide Box Set)             (Skaven Paint Set)     0.20    0.666667

Results returned in  0.002022266387939453 seconds from mlxtend fpgrowth implementation
PS C:\Users\ozdem\Desktop\CS 634\midterm_repo\cs634_midterm_proj>
```

Usage Notes

For this project to function properly, the installation of the pandas and mlxtend Python libraries are required.

You may install these libraries using the following commands:

pip install pandas

pip install mlxtend

The version of Python used for this project is version 3.12.6.

Data sets are provided, but if other data sets are to be used they must follow a specific format detailed below.

The project expects the .csv files to match the following format, as illustrated below.

1	Item #	Item Name
2	1	Dragon Age: The Veilguard
3	2	Weird West
4	3	Another Crab's Treasure
5	4	Dualsense Controller
6	5	Joy-Con
7	6	Xbox Wireless Controller
8	7	Wireless Headset
9	8	Memory Card
10	9	Controller Charging Station
11	10	Elden Ring

Figure: gamestop.csv

Each item name is contained in a column called “Item Name”, and each item takes up one row. “Item Name” is case sensitive and must be replicated exactly to avoid calculation errors. The names of items are also case sensitive and must appear in this .csv file exactly as they appear in each transaction, so “memory card” and “Memory Card” are NOT considered the same item.

1	Transaction ID	Transaction
2	T1	Dragon Age: The Veilguard, Weird West, Dualsense Controller, Wireless Headset
3	T2	Weird West, Controller Charging Station, Dualsense Controller
4	T3	Joy-Con, Memory Card, Another Crab's Treasure
5	T4	Wireless headset, Memory Card, Controller Charging Station
6	T5	Elden Ring, Another Crab's Treasure, Dualsense Controller
7	T6	Dragon Age: The Veilguard, Wireless Headset, Controller Charging Station
8	T7	Memory Card, Joy-Con, Wireless Headset
9	T8	Another Crab's Treasure, Elden Ring, Dualsense Controller
10	T9	Xbox Wireless Controller, Controller Charging Station, Wireless Headset, Elden Ring
11	T10	Weird West, Dualsense Controller, Wireless Headset, Memory Card
12	T11	Another Crab's Treasure, Elden Ring, Wireless Headset
13	T12	Another Crab's Treasure, Weird West
14	T13	Elden Ring, Dualsense Controller, Controller Charging Station
15	T14	Memory Card, Joy-Con, Controller Charging Station, Another Crab's Treasure
16	T15	Dragon Age: The Veilguard, Dualsense Controller, Wireless Headest, Memory Card
17	T16	Dragon Age: The Veilguard, Another Crab's Treasure, Weird West
18	T17	Elden Ring, Weird West, Dualsense Controller
19	T18	Dragon Age: The Veilguard, Xbox Wireless Controller, Wireless Headset
20	T19	Weird West, Dualsense Controller
21	T20	Another Crab's Treasure, Weird West

Figure: gamestop_transactions.csv

Every transaction is recorded in a column called “Transactions”, and a transaction is contained in a cell with each item name in the transaction delimited by a comma. One row represents a single transaction. Once again, every value is case sensitive.

All .csv files are expected to be in the same directory as the .py project file for the program to work.

These datasets were generated arbitrarily by hand with an arbitrary selection of items chosen from each retailer’s online storefronts. Item names were simplified for clarity.

Other Notes

The github repository for this project follows:

https://github.com/atwoodmachine/cs634_midterm_proj

It contains all used .csv files, the Python source code, and a Jupyter Notebook for a demonstration of output, as well as a copy of this report.