Bài tập thực hành-Khai thác dữ liệu-tuần 4

Phan Hồng Trâm - 21110414

April 2024

Mục lục

1 trình bày tóm tắt lại phần code

- Import thư viện:

```
import itertools
```

  - Thư viện này cung cấp các hàm để lặp hiệu quả và tạo ra các kết hợp phần tử cụ thể từ các lần lặp.

- `def generateC1(dataSet)`: Hàm này khởi tạo tập ứng cử viên ban đầu (C1) từ dataset, là danh sách các giao dịch (transactions).

```
def generateC1(dataSet):
    productDict = {}
    returneSet = []
    for data in dataSet:
        for product in data:
            if product not in productDict:
               productDict[product] = 1
            else:
                 productDict[product] = productDict[product] + 1
    for key in productDict:
        tempArray = []
        tempArray.append(key)
        returneSet.append(tempArray)
        returneSet.append(productDict[key])
        tempArray = []
    return returneSet
```

  - `productDict`: từ điển trống để lưu trữ tần suất của từng mục.
  - `for data in dataSet`: Lặp qua từng transaction (`data`) trong tập dữ liệu
    - Đối với từng item (`product`) trong `data`:
      - Nếu mục này chưa có trong `ProductDict`, nó sẽ thêm mục đó làm khóa (key) có giá trị là 1 (số lượng ban đầu của nó).
      - Nếu mục này đã có sẵn, nó sẽ tăng giá trị được liên kết với khóa đó trong từ điển lên 1.
  - Sau khi lặp qua tất cả các giao dịch, nó tạo ra một danh sách mới `returneSet`.
  - Sau đó lặp qua từng `key` trong `ProductDict`:
    - Tạo một danh sách tạm thời `tempArray` và thêm item (`key`) hiện tại vào đó.
    - Thêm `tempArray` (chỉ chứa 1 item duy nhất) và `productDict[key]` từ từ điển vào `returneSet`.
    - Xóa `tempArray` để cho lần lặp tiếp theo.
  - Hàm trả về `returneSet` chứa danh sách 1-item đại diện cho tập ứng cử viên (candidate) ban đầu C1.

- `def generateFrequentItemSet`: Hàm này tạo các tập mục thường xuyên bằng cách lấy các tập ứng cử viên làm đầu vào.

```
def generateFrequentItemSet(CandidateList, noOfTransactions, minimumSupport, dataSet,
    fatherFrequentArray):
    frequentItemsArray = []
    for i in range(len(CandidateList)):
        if i%2 != 0:
            support = (CandidateList[i] * 1.0 / noOfTransactions) * 100
            if support >= minimumSupport:
```

```
                frequentItemsArray.append(CandidateList[i-1])
                frequentItemsArray.append(CandidateList[i])
            else:
                eleminatedItemsArray.append(CandidateList[i-1])

    for k in frequentItemsArray:
        fatherFrequentArray.append(k)

    if len(frequentItemsArray) == 2 or len(frequentItemsArray) == 0:
        returnArray = fatherFrequentArray
        return returnArray

    else:
        generateCandidateSets(dataSet, eleminatedItemsArray, frequentItemsArray, noOfTransactions,
    minimumSupport)
```

- Hàm này nhận các đối số:
  - `CandidateList`: Danh sách các tập mục ứng cử viên.
  - `noOfTransactions`: Tổng số giao dịch trong tập dữ liệu.
  - `minimumSupport`: Độ phổ biến tối thiểu (phần trăm).
  - `dataSet`: Tập dữ liệu gốc chứa các giao dịch.
  - `fatherFrequentArray`: Ban đầu là một danh sách trống, được sử dụng để lưu trữ frequent itemsets.
- `frequentItemsArray`: Danh sách trống dùng để lưu các frequent itemsets.
- `for i in range(len(CandidateList))`: Bắt đầu lặp qua `CandidateList`:
  - `if i%2 != 0`: Kiểm tra xem `i` có lẻ không. Nếu có, ta tính toán độ phổ biến của itemset hiện tại bằng cách chia `CandidateList[i]` cho tổng số giao dịch (`noOfTransactions`) và nhân với 100 để biểu thị dưới dạng phần trăm.
    - `if support>=minimumSupport`: Nếu độ phổ biến lớn hơn hoặc bằng độ phổ biến tối thiểu thì ta thêm tập itemset từ chỉ mục chẵn trước đó `CandidateList[i-1]` vào `frequentItemsArray`. Sau đó ta cũng thêm tập itemset hiện tại `CandidateList[i]` vào `frequentItemsArray`.
    - `else`: Ngược lại (nếu độ phổ biến nhỏ hơn mức tối thiểu), nó sẽ coi tập mục này không thường xuyên và không thêm nó.
- Sau khi duyệt qua danh sách ứng cử viên, nó sẽ thêm tất cả các tập phổ biến được tìm thấy vào `parentFrequentArray`.
- `if len(frequentItemsArray) == 2 or len(frequentItemsArray) == 0`: Nếu độ dài là 2 (nghĩa là chỉ tìm thấy một tập mục phổ biến) hoặc 0 (không tìm thấy tập mục phổ biến nào), thì nó coi đây là trường hợp cơ sở và trả về `parentFrequentArray` chứa các tập mục phổ biến cuối cùng.
- `else`: Mặt khác nó gọi đệ quy hàm `generateCandidateSets` nếu vẫn còn các mục thường xuyên và xây dựng mảng tập mục phổ biến cuối cùng.

- `def generateCandidateSets`: Hàm này tạo tập các ứng viên bằng cách nhận các nhóm thường xuyên (`FrequentItemSet`) làm đầu vào.

```
def generateCandidateSets(dataSet, eleminatedItemsArray, frequentItemsArray, noOfTransactions,
    minimumSupport):
    onlyElements = []
    arrayAfterCombinations = []
    candidateSetArray = []
    for i in range(len(frequentItemsArray)):
        if i%2 == 0:
            onlyElements.append(frequentItemsArray[i])
```

```
    for item in onlyElements:
        tempCombinationArray = []
        k = onlyElements.index(item)
        for i in range(k + 1, len(onlyElements)):
            for j in item:
                if j not in tempCombinationArray:
                    tempCombinationArray.append(j)
            for m in onlyElements[i]:
                if m not in tempCombinationArray:
                    tempCombinationArray.append(m)
            arrayAfterCombinations.append(tempCombinationArray)
            tempCombinationArray = []
    sortedCombinationArray = []
    uniqueCombinationArray = []
    for i in arrayAfterCombinations:
        sortedCombinationArray.append(sorted(i))
    for i in sortedCombinationArray:
        if i not in uniqueCombinationArray:
            uniqueCombinationArray.append(i)
    arrayAfterCombinations = uniqueCombinationArray
    for item in arrayAfterCombinations:
        count = 0
        for transaction in dataSet:
            if set(item).issubset(set(transaction)):
                count = count + 1
        if count != 0:
            candidateSetArray.append(item)
            candidateSetArray.append(count)
    generateFrequentItemSet(candidateSetArray, noOfTransactions, minimumSupport, dataSet,
    fatherFrequentArray)
```

- Hàm này nhận các đối số tương tự như `generateFrequentItemSet`:
  - `dataSet`: Tập dữ liệu gốc chứa các giao dịch.
  - `eleminatedItemsArray`: Danh sách rỗng ban đầu, được sử dụng để lưu trữ các tập bị loại bỏ (không sử dụng trong phần triển khai này).
  - `frequentItemsArray`: Danh sách các tập thường xuyên từ hàm trước.
- Khởi tạo:
  - `onlyElements`: Danh sách này sẽ lưu trữ các mặt hàng riêng lẻ từ các tập thường xuyên được tìm thấy trong bước trước (generateFrequentItemSet).
  - `arrayAfterCombinations`: Danh sách này sẽ tạm thời lưu trữ tất cả các kết hợp có thể có của các mặt hàng được tạo từ danh sách onlyElements.
  - `candidateSetArray`: Danh sách này sẽ lưu trữ các tập ứng cử viên cuối cùng cùng với số đếm hỗ trợ của chúng.
- `for i in range(len(frequentItemsArray))`: lặp qua các tập thường xuyên:
  - `if i%2 == 0`: nó kiểm tra xem chỉ mục hiện tại `i` có chẵn hay không. Nếu có, ta trích xuất tập thường xuyên hiện tại và thêm nó vào `onlyElements`.
- Sau khi tạo danh sách `onlyElements`, ta lặp qua mỗi phần tử (`item`) trong danh sách này:
  - tạo một danh sách tạm thời `tempCombinationArray` để lưu trữ sự kết hợp hiện tại.
  - Nó lấy chỉ mục `k` bằng phần tử hiện tại (`item`) trong danh sách `onlyElements`.
  - `for i in range(k + 1, len(onlyElements))`: Lặp qua tất cả các phần tử `i` trong danh sách `onlyElements` bắt đầu từ phần tử sau phần tử hiện tại `k + 1`.

  - Bên trong vòng lặp lồng nhau này, nó lặp qua mỗi phần tử `j` trong phần tử hiện tại `item`: Nếu `j` chưa có trong `tempCombinationArray`, ta thêm `j` vào `tempCombinationArray`
  - Sau đó, nó lặp qua mỗi phần tử `m` trong phần tử tiếp theo `onlyElements[i]` từ danh sách `onlyElements`: Tương tự như vòng lặp trước, nó thêm phần tử `m` vào `tempCombinationArray` chỉ khi nó chưa có trong danh sách.
  - Khi kết hợp cho phần tử hiện tại được xây dựng, nó thêm `tempCombinationArray` (chứa kết hợp hiện tại) vào danh sách `arrayAfterCombinations`.
  - xóa `tempCombinationArray` cho sự kết hợp tiếp theo.
- Bây giờ, nó đã tạo ra tất cả các kết hợp ứng cử viên có thể trong `arrayAfterCombinations`. Ta tạo hai danh sách khác:
  - `sortedCombinationArray`: Danh sách này sẽ lưu trữ tất cả các kết hợp từ `arrayAfterCombinations` nhưng được sắp xếp để dễ so sánh hơn.
  - `uniqueCombinationArray`: Danh sách này sẽ lưu trữ chỉ các kết hợp duy nhất (không trùng lặp).
- Tiếp theo ta lặp qua mỗi **kết hợp** `i` trong `arrayAfterCombinations`:
  - Ta tạo phiên bản được sắp xếp của kết hợp và thêm nó vào `sortedCombinationArray`.
- Ta lặp qua các **kết hợp được sắp xếp** `i` trong `sortedCombinationArray`:
  - Nếu kết hợp hiện tại `i` chưa có trong `uniqueCombinationArray`, ta thêm nó vào danh sách.
- Bây giờ, `uniqueCombinationArray` chứa tất cả các tập ứng cử viên duy nhất và được sắp xếp. Nó thay thế danh sách `arrayAfterCombinations` bằng phiên bản duy nhất này.
- Tiếp theo ta tính toán số đếm hỗ trợ cho mỗi tập ứng cử viên trong `uniqueCombinationArray`:
  - Bên trong vòng lặp lồng nhau, nó chuyển đổi cả tập ứng cử viên (`item`) và giao dịch (`transaction`) thành bộ bằng cách sử dụng `set()`. Sau đó, nó kiểm tra xem tất cả các phần tử từ tập ứng cử viên (`set(item)`) có phải là một tập con của giao dịch (`set(transaction)`) hay không bằng cách sử dụng phương thức `issubset`. Nếu có, nó tăng bộ đếm `count` lên 1.
  - Sau khi lặp qua tất cả các giao dịch, nếu `count` lớn hơn 0 (nghĩa là tập ứng cử viên xuất hiện trong ít nhất một giao dịch), nó thêm tập ứng cử viên (`item`) và số đếm hỗ trợ của nó (`count`) vào `candidateSetArray`.
- Cuối cùng hàm này gọi đệ quy lại hàm `generateFrequentItemSet` với tập ứng cử viên mới.

- `def generateAssociationRule(freqSet):`

```
def generateAssociationRule(freqSet):
    associationRule = []
    for item in freqSet:
        if isinstance(item, list):
            if len(item) != 0:
                length = len(item) - 1
                while length > 0:
                    combinations = list(itertools.combinations(item, length))
                    temp = []
                    LHS = []
                    for RHS in combinations:
                        LHS = set(item) - set(RHS)
                        temp.append(list(LHS))
                        temp.append(list(RHS))
                        #print(temp)
                        associationRule.append(temp)
                        temp = []
                    length = length - 1
    return associationRule
```

- Hàm này nhận một danh sách các tập thường xuyên `freqSet` làm đầu vào.
- Nó tạo một danh sách rỗng `associationRule` để lưu trữ các quy tắc liên kết được tạo.
- `for item in freqSet`: Lặp qua mỗi `item` trong `freqSet`:
  * `if isinstance(item, list)`: kiểm tra xem mục hiện tại (`item`) có phải là danh sách (nghĩa là tập thường xuyên) và nó không rỗng.
    · Nếu cả hai điều kiện được đáp ứng, nó tính toán độ dài `length` của tập thường xuyên hiện tại `item` bằng cách lấy `len(item)-1`. Điều này đại diện cho tổng số mục trong tập thường xuyên.
    · `while length > 0`: Vòng lặp này tạo ra tất cả các kết hợp có thể có của các mục (quy tắc liên kết) từ tập thường xuyên hiện tại.
    · Bên trong vòng lặp, nó sử dụng chức năng `itertools.combinations` từ thư viện `itertools` để tạo tất cả các kết hợp của `length` mục từ `item` (tập thường xuyên). Kết quả được chuyển đổi thành danh sách bằng cách sử dụng `list()`. Điều này tạo ra tất cả các kết hợp `LHS` (Left-Hand Side) và `RHS` (Right-Hand Side) cho các quy tắc liên kết.
    · Tạo hai danh sách tạm thời:
    · `temp`: Danh sách này sẽ tạm thời lưu trữ LHS và RHS hiện tại của quy tắc liên kết được tạo.
    · `LHS`: Danh sách này sẽ lưu trữ Bên trái (LHS) của quy tắc liên kết cho lần lặp hiện tại.
    · `for RHS in combinations`: Nó tính toán `LHS` (Bên trái) của quy tắc liên kết bằng cách tìm sự khác biệt giữa tập các mục trong tập thường xuyên hiện tại (`item`) và kết hợp hiện tại (`RHS`) bằng cách sử dụng các thao tác của tập `set()`. Nó chuyển đổi cả hai thành tập bằng cách sử dụng `set()` và sử dụng toán tử − để tìm sự khác biệt. Sau đó ta thêm `list(LHS)` và `list(RHS)` vào danh sách `temp`. Rồi ta thêm toàn bộ danh sách `temp` (chứa LHS và RHS) vào danh sách `associationRule`.
    · xóa danh sách `temp` cho lần lặp tiếp theo.
    · Sau khi lặp qua tất cả các kết hợp có thể (`RHS`) cho `length` hiện tại, nó giảm `length` xuống 1 để tạo ra các kết hợp với ít hơn một mục trong lần lặp tiếp theo.
- hàm trả về danh sách `associationRule` chứa tất cả các quy tắc liên kết được tạo.

- `def apriori0utput(rules, dataSet, minimumSupport, minimumConfidence):`

```
def apriori0utput(rules, dataSet, minimumSupport, minimumConfidence):
    returnApriori0utput = []
    for rule in rules:
        supportOfX = 0
        supportOfXinPercentage = 0
        supportOfXandY = 0
        supportOfXandYinPercentage = 0
        for transaction in dataSet:
            if set(rule[0]).issubset(set(transaction)):
                supportOfX = supportOfX + 1
            if set(rule[0] + rule[1]).issubset(set(transaction)):
                supportOfXandY = supportOfXandY + 1
        supportOfXinPercentage = (supportOfX * 1.0 / noOfTransactions) * 100
        supportOfXandYinPercentage = (supportOfXandY * 1.0 / noOfTransactions) * 100
        confidence = (supportOfXandYinPercentage / supportOfXinPercentage) * 100
        if confidence >= minimumConfidence:
            supportOfXAppendString = "Support Of X: " + str(round(supportOfXinPercentage, 4))
            supportOfXandYAppendString = "Support of X & Y: " + str(round(supportOfXandYinPercentage
    , 4))
            confidenceAppendString = "Confidence: " + str(round(confidence))

            returnApriori0utput.append(supportOfXAppendString)
            returnApriori0utput.append(supportOfXandYAppendString)
```

```
            returnAprioriOutput.append(confidenceAppendString)
            returnAprioriOutput.append(rule)

    return returnAprioriOutput
```

– Hàm này nhận nhiều đối số:

  * `rules`: Danh sách các quy tắc liên kết được tạo bởi hàm generateAssociationRule.
  * `dataSet`: Tập dữ liệu gốc chứa các giao dịch.
  * `minimumSupport`: Độ phổ biến tối thiểu (tỷ lệ phần trăm).
  * `minimumConfidence`: Độ tin cậy tối thiểu (tỷ lệ phần trăm).

– Nó tạo một danh sách rỗng `returnAprioriOutput` để lưu trữ đầu ra cuối cùng với độ phổ biến và độ tin cậy được tính toán.

– Nó lặp qua mỗi quy tắc `rule` trong danh sách `rules` (chứa `LHS` và `RHS` của các quy tắc liên kết).

– Khởi tạo một số biến:

  * `supportOfX`: Biến này sẽ lưu trữ số đếm độ phổ biến cho `LHS(X)` của quy tắc liên kết hiện tại.
  * `supportOfXinPercentage`: Biến này sẽ lưu trữ số đếm hỗ trợ cho `LHS(X)` dưới dạng tỷ lệ phần trăm.
  * `supportOfXandY`: Biến này sẽ lưu trữ số đếm độ phổ biến cho toàn bộ quy tắc (X và Y).
  * `supportOfXandYinPercentage`: Biến này sẽ lưu trữ số đếm độ phổ biến cho toàn bộ quy tắc (X và Y) dưới dạng tỷ lệ phần trăm.
  * `confidence`: Biến này sẽ lưu trữ độ tin cậy của quy tắc liên kết hiện tại.

– Nó lặp qua mỗi giao dịch `transaction` trong tập dữ liệu `dataSet`:

  * Bên trong vòng lặp lồng nhau này, nó chuyển đổi cả `LHS` của quy tắc hiện tại `rule[0]` và giao dịch hiện tại `transaction` thành tập bằng cách sử dụng `set()`. Sau đó, nó kiểm tra xem tất cả các phần tử từ `LHS` (`set(rule[0])`) có phải là một tập con của giao dịch `set(transaction)` hay không bằng cách sử dụng phương thức `issubset`.
  * Nếu tất cả các phần tử có mặt (một tập con), nó tăng `supportOfX` (số đếm độ phổ biến cho `LHS`).
  * Nó tính toán số đếm phổ biến cho toàn bộ quy tắc (X và Y) bằng cách chuyển đổi cả quy tắc toàn bộ `rule[0] + rule[1]` và giao dịch `transaction` thành tập và sử dụng phương thức `issubset` tương tự như bước trước. Nếu tất cả các phần tử có mặt, nó tăng `supportOfXandY`.
  * Sau khi lặp qua tất cả các giao dịch, nó tính toán độ phổ biến của `LHS` (`supportOfX`) và toàn bộ quy tắc (`supportOfXandY`) dưới dạng tỷ lệ phần trăm bằng cách chia số đếm của chúng cho tổng số giao dịch (`noOfTransactions`) và nhân với 100.
  * Tính toán độ tin cậy `confidence` của quy tắc liên kết hiện tại bằng cách sử dụng công thức: `confidence = (support of X and Y / support of X) * 100`

* Thay thế các vị trí giữ chỗ bằng các giá trị được tính toán trước đó:
  `confidence = (supportOfXandYinPercentage / supportOfXinPercentage) * 100`
* Kiểm tra xem độ tin cậy `confidence` có lớn hơn hoặc bằng ngưỡng độ tin cậy tối thiểu `minimumConfidence` do người dùng chỉ định hay không.
  - Nếu độ tin cậy đáp ứng ngưỡng tối thiểu, ta tiến hành tạo các chuỗi được định dạng cho độ phổ biến của `LHS` (`supportOfXAppendString`), hỗ trợ của toàn bộ quy tắc (`supportOfXandYAppendString`), và độ tin cậy (`confidenceAppendString`). Các chuỗi này sẽ được sử dụng cho đầu ra cuối cùng.
  - Nếu không (nếu độ tin cậy thấp hơn mức tối thiểu), nó chuyển sang quy tắc tiếp theo trong vòng lặp `rules`.
  - Cuối cùng, hàm thêm tất cả các chuỗi được định dạng `supportOfXAppendString`, `supportOfXandYAppendString`, `confidenceAppendString`, và quy tắc thực tế `rule` vào danh sách `returnAprioriOutput`.

– trả về danh sách các quy tắc liên kết `rules`

- Khởi tạo các biến và chạy chương trình:

```
minimumSupport = input('Enter minimum Support in percentage: ')
minimumConfidence = input('Enter minimum Confidence in percentage: ')
fileName = "data.txt"


minimumSupport = int(minimumSupport)
minimumConfidence = int(minimumConfidence)

nonFrequentSets = []
allFrequentItemSets = []
tempFrequentItemSets = []
dataSet = []
eleminatedItemsArray = []
noOfTransactions = 0
fatherFrequentArray = []

#   Reading the data file line by line
with open(fileName,'r') as fp:
    lines = fp.readlines()

for line in lines:
    line = line.rstrip()
    dataSet.append(line.split(","))

noOfTransactions = len(dataSet)

firstCandidateSet = generateC1(dataSet)

frequentItemSet = generateFrequentItemSet(firstCandidateSet, noOfTransactions, minimumSupport,
    dataSet, fatherFrequentArray)

associationRules = generateAssociationRule(fatherFrequentArray)

AprioriOutput = aprioriOutput(associationRules, dataSet, minimumSupport, minimumConfidence)


counter = 1
if len(AprioriOutput) == 0:
```

```
    print("There are no association rules for this support and confidence.")
else:
    for i in AprioriOutput:
        if counter == 4:
            print(str(i[0]) + "--->" + str(i[1]))
            counter = 0
        else:
            print(i, end='  ')
        counter = counter + 1
```

- `nonFrequentSets`: Danh sách rỗng (không sử dụng trong mã được cung cấp).
- `allFrequentItemSets`: Danh sách rỗng để lưu trữ tất cả các tập thường xuyên được tìm thấy.
- `tempFrequentItemSets`: Danh sách rỗng (không sử dụng trong mã được cung cấp).
- `dataSet`: Danh sách rỗng để lưu trữ các giao dịch đọc từ tệp.
- `eleminatedItemsArray`: Danh sách rỗng (không sử dụng trong mã được cung cấp).
- `noOfTransactions`: Biến số nguyên được khởi tạo bằng 0 để lưu trữ tổng số giao dịch.
- `fatherFrequentArray`: Danh sách rỗng để lưu trữ các tập thường xuyên từ các lần lặp trước.
- Mở tệp dữ liệu `fileName` ở chế độ đọc `'r'` bằng `with open()`.
- Đọc tất cả các dòng từ tệp bằng `readlines()` và lưu trữ chúng trong danh sách `lines`.
- Lặp qua từng dòng `line` trong danh sách `lines`: Loại bỏ khoảng trắng thừa ở cuối dòng bằng `rstrip()`. Chia dòng thành danh sách các mục (giao dịch) bằng dấu , làm dấu phân cách và thêm nó vào `dataSet`.
- Tính toán tổng số giao dịch `noOfTransactions` bằng cách tìm độ dài của `dataSet`.
- Ta gọi hàm:

```
noOfTransactions = len(dataSet)

firstCandidateSet = generateC1(dataSet)

frequentItemSet = generateFrequentItemSet(firstCandidateSet, noOfTransactions, minimumSupport,
    dataSet, fatherFrequentArray)

associationRules = generateAssociationRule(fatherFrequentArray)

AprioriOutput = aprioriOutput(associationRules, dataSet, minimumSupport, minimumConfidence)
```

- Cuối cùng ta kiểm tra xem có tìm thấy luật kết hợp nào không. Nếu không, nó sẽ hiển thị thông báo cho biết không có quy tắc nào đáp ứng tiêu chí. Nếu có, ta in ra kết quả:

```
counter = 1
if len(AprioriOutput) == 0:
    print("There are no association rules for this support and confidence.")
else:
    for i in AprioriOutput:
        if counter == 4:
            print(str(i[0]) + "--->" + str(i[1]))
            counter = 0
        else:
            print(i, end='  ')
        counter = counter + 1
```

2 So sánh kết quả với hàm có sẵn trong thư viện

- Chạy thử kết quả: Với minimum Support là 60% và minimum Confidence là 10%:

```
Enter minimum Support in percentage: 60
Enter minimum Confidence in percentage: 10
Support Of X: 72.7273  Support of X & Y: 63.6364  Confidence: 87  ['Wine']------>['Milk']
Support Of X: 77.2727  Support of X & Y: 63.6364  Confidence: 82  ['Milk']------>['Wine']
PS C:\Users\PC\Desktop\CODE\PY\Năm 3 HKII\KPDL\lab04>
```

Hình 1: Kết quả của thuật toán Apriori

- Sau khi chạy chương trình, ta được kết quả Support là 63.6364 và 63.6364 trùng với kết quả của hàm có sẵn trong thư viện.

	antecedents	consequents	support
0	(Milk)	(Wine)	0.636364
1	(Wine)	(Milk)	0.636364

Hình 2: Kết quả của thuật toán Apriori sử dụng thư viện có sẵn