Apriori Algorithm

DWDM Lab 7 - 180911202

Nikesh Kumar

May 29, 2021

Contents

1 Apriori Algorithm

1.1 Algorithm Implementation

[1]:
```
import numpy as np
import pandas as pd
import itertools
import io
import os
import sys
from pprint import pprint
```

[2]:
```
# Generating candidate set(l+1) from frequent set(l)
def generate(frequent_set: list):
    candidate_set = list()
    for itemset_1 in frequent_set:
        for itemset_2 in frequent_set:
            if type(itemset_1) == str:
                itemset_1 = [itemset_1]
            if type(itemset_2) == str:
                itemset_2 = [itemset_2]
            set_1 = list(sorted(itemset_1))
            set_2 = list(sorted(itemset_2))
            if set_1 != set_2 and set_1[:-1] == set_2[:-1] and set_1[-1] < set_2[-1]:
                candidate_set.append(list(sorted(set(set_1)|set(set_2))))
    return sorted(candidate_set)

# Finding subsets of size l in an itemset in candidate set(l+1) for pruning
def findsubsets(itemset: list, size: int):
    return [subset for subset in itertools.combinations(itemset, size)]

# Removing itemsets from candidate set that are not present in frequent set
def prune(candidate_set: list,frequent_set: list,idx: int):
    for itemset in candidate_set:
        subsets = findsubsets(itemset,idx - 1)
        for subset in subsets:
            if subset in frequent_set:
                candidate_set.pop(subset)
                break
    return candidate_set

# Counts support for each itemset and returns a frequent itemset
def support_counter(candidate_set: list):
    frequent_set = dict()
```

```
    for itemset in candidate_set:
        for transaction in transactions:
            if (set(itemset)<=set(transaction)):
                key_code = ",".join(itemset)
                if key_code in frequent_set:
                    frequent_set[key_code] += 1
                else:
                    frequent_set[key_code] = 1
    return frequent_set

# Finds confidence and support for an association rule
def find_conf_support(association_left: list,association_right: list):
    frequent_set = dict()
    itemset = list(sorted(set(association_left)|set(association_right)))
    association_count = 0
    total_count = 0
    for transaction in transactions:
        if set(itemset) <= set(transaction):
            association_count += 1
        if set(association_left) <= set(transaction):
            total_count += 1
    try:
        confidence = association_count/total_count
        support = association_count/len(transactions)
    except ZeroDivisionError:
        confidence = 0
        support = 0
    return {"support": support,"confidence": confidence}

# Finds frequency of each itemset in candidate set and then returns the frequent␣
 ↪itemset
def transaction_check(candidate_set: list, support_count: int):
    frequent_set = support_counter(candidate_set)
    filtered_itemset = list(filter(lambda x: frequent_set[x] >␣
 ↪support_count,frequent_set))
    return list(map(lambda x: x.split(','),filtered_itemset))

# Apriori Algorithm
def apriori(support_count: int, supress_output : bool = False):
    save_output = sys.stdout
    sys.stdout = (sys.stdout,open(os.devnull, 'w',␣
 ↪encoding='utf-8'))[supress_output]
    candidate_sets = list()
    frequent_sets = list()
    idx = 2

    initial_set = dict()
```

```
    for transaction in transactions:
        for item in transaction:
            if item in initial_set:
                initial_set[item] += 1
            else:
                initial_set[item] = 1

    candidate_sets.append([])
    frequent_sets.append([])
    candidate_sets.append([item for item in initial_set.keys()])
    filtered_set = list(filter(lambda x: initial_set[x] > support_count,initial_set))
    frequent_sets.append([[item] for item in filtered_set])

    while True:
        candidate_sets.append(generate(frequent_sets[idx-1]))
        candidate_sets[idx] = prune(candidate_sets[idx],frequent_sets[idx-1],idx)
        frequent_set = transaction_check(candidate_sets[idx],support_count)
        if len(frequent_set) != 0:
            frequent_sets.append(frequent_set)
        else:
            break
        print("----------------------------------------------------------------------\n")
        print("Round {}:\n\n\tFrequent Itemsets: {}".format(idx,frequent_sets[idx]))
        print("\n----------------------------------------------------------------------")
        idx += 1
    sys.stdout = save_output
    return frequent_sets

# Analyses the number of frequent itemsets and the maximal sets encountered
def frequent_item_analysis(frequent_sets: list,support_count: int):
    i = 0
    initial_set = dict()
    for transaction in transactions:
        for item in transaction:
            if item in initial_set:
                initial_set[item] += 1
            else:
                initial_set[item] = 1
    iter_list = frequent_sets[1:]
    for frequent_set in iter_list:
        __itemset_len = len(frequent_set)
        __support_dict = support_counter(frequent_set)
        if i == 0:
```

```
            __support_dict = initial_set
        __max_item = max(__support_dict,key=lambda x: __support_dict[x])
        __max_count = __support_dict[__max_item]
        print("{}-Itemsets({}) → ({}):{}".format(i+1,__itemset_len,__max_item,__max_count))
        i += 1
    print("Total number of transactions: ",len(transactions))

# Finds all association rules for a given itemset
def association_rules(frequent_set: list,min_support : float = 0.0,min_confidence : float = 0.0):
    size = len(frequent_set)
    for __size in range(1,size):
        subsets = findsubsets(frequent_set,__size)
        for subset in subsets:
            absent_subset = set(frequent_set) - set(subset)
            stats = find_conf_support(list(subset),list(absent_subset))
            if (stats["confidence"] < min_confidence) or (stats["support"] < min_support):
                continue
            print("\033[93m{:^35} → {:^35}\033[0m\n\033[91m{:^70}\033[0m\n".format(','.join(subset),','.join(absent_subset),str(stats)))

# Printing all possible associations for a given
def print_all_rules(frequent_itemsets: list,min_support : float = 0.0,min_confidence : float = 0.0):
    for idx,frequent_itemset in enumerate(frequent_itemsets):
        for itemset in frequent_itemset:
            association_rules(itemset,min_support,min_confidence)
```

1.2 Dataset I: Groccery Market Basket Analysis

https://www.kaggle.com/irfanasrullah/groceries

1.2.1 Processing Dataset

[3]:
```python
# Dataset from https://www.kaggle.com/irfanasrullah/groceries

dataset = pd.read_csv('./groceries.csv')
dataset.head()
```

[3]:
```
   Item(s)            Item 1               Item 2           Item 3       ...
0        4      citrus fruit  semi-finished bread        margarine       ...
1        3    tropical fruit               yogurt           coffee       ...
2        1        whole milk                  NaN              NaN       ...
3        4         pip fruit               yogurt     cream cheese       ...
4        4  other vegetables           whole milk  condensed milk       ...

[5 rows x 33 columns]
```

[4]:
```python
transactions = list()
for row in range(len(dataset)):
    transactions.append(list(dataset.iloc[row,1:].dropna()))

pprint(transactions[:3])
```

```
[['citrus fruit', 'semi-finished bread', 'margarine', 'ready soups'],
 ['tropical fruit', 'yogurt', 'coffee'],
 ['whole milk']]
```

1.2.2 Running the Apriori Algorithm

[5]:
```python
support_count = int(input("Enter the minimum support count threshold: "))
print("Minimum Support Count: ", support_count)
frequent_itemsets = apriori(support_count,supress_output = True)
```

```
Enter the minimum support count threshold: 34
Minimum Support Count:  34
```

1.2.3 Frequent Sets with maximum frequency

[6]:
```
frequent_item_analysis(frequent_itemsets,support_count)
```

```
1-Itemsets(130) → (whole milk):2513
2-Itemsets(956) → (other vegetables,whole milk):736
3-Itemsets(603) → (other vegetables,root vegetables,whole milk):228
4-Itemsets(57) → (other vegetables,root vegetables,whole milk,yogurt):77
5-Itemsets(1) → (other vegetables,root vegetables,tropical fruit,whole
milk,yogurt):35
Total number of transactions:  9835
```

1.2.4 Finding Associations for a given support and confidence

[7]:
```
min_support = float(input("Enter min support threshold: "))
min_confidence = float(input("Enter min confidence threshold: "))
print_all_rules(frequent_itemsets,min_support = min_support,min_confidence =␣
 ↪min_confidence)
```

```
Enter min support threshold: 0.02
Enter min confidence threshold: 0.4
              beef                    →              whole milk
  {'support': 0.02125063548551093, 'confidence': 0.4050387596899225}

             butter                   →              whole milk
  {'support': 0.02755465175394001, 'confidence': 0.4972477064220184}

              curd                    →              whole milk
 {'support': 0.026131164209456024, 'confidence': 0.4904580152671756}

          domestic eggs               →              whole milk
 {'support': 0.029994916115912557, 'confidence': 0.4727564102564102 4}

        frozen vegetables             →              whole milk
  {'support': 0.02043721403152008, 'confidence': 0.4249471458773784}

            margarine                 →              whole milk
 {'support': 0.024199288256227757, 'confidence': 0.41319444444444444}

         root vegetables              →           other vegetables
 {'support': 0.047381799694966954, 'confidence': 0.43470149253731344}

       whipped/sour cream             →           other vegetables
 {'support': 0.02887646161667514, 'confidence': 0.40283687943262414}

         root vegetables              →              whole milk
 {'support': 0.048906964921199794, 'confidence': 0.44869402985074625}
```

```
        tropical fruit              →              whole milk
{'support': 0.04229791560752415, 'confidence': 0.40310077519379844}

       whipped/sour cream           →              whole milk
{'support': 0.032231825114387394, 'confidence': 0.44964539007092197}

            yogurt                  →              whole milk
{'support': 0.05602440264361973, 'confidence': 0.40160349854227406}

other vegetables,root vegetables    →              whole milk
{'support': 0.023182511438739197, 'confidence': 0.4892703862660944}

   root vegetables,whole milk       →           other vegetables
{'support': 0.023182511438739197, 'confidence': 0.47401247401247404}

     other vegetables,yogurt        →              whole milk
 {'support': 0.022267412302999492, 'confidence': 0.5128805620608899}
```

1.2.5 Mining Rules for a given itemset input

[8]:
```python
frequent_set = list(map(lambda x: str(x).strip(),input("Enter an itemset␣
 ↪separated by commas: ").split(",")))
print("\033[1mMining rules for:",frequent_set,"\033[0m")
association_rules(frequent_set)
```

```
Enter an itemset separated by commas: whole milk, yogurt, rolls/buns
Mining rules for: ['whole milk', 'yogurt', 'rolls/buns']
            whole milk              →           yogurt,rolls/buns
{'support': 0.015556685307574987, 'confidence': 0.060883406287306006}

              yogurt                →         rolls/buns,whole milk
 {'support': 0.015556685307574987, 'confidence': 0.11151603498542274}

            rolls/buns              →           yogurt,whole milk
 {'support': 0.015556685307574987, 'confidence': 0.0845771144278607}

         whole milk,yogurt          →              rolls/buns
 {'support': 0.015556685307574987, 'confidence': 0.2776769509981851}

       whole milk,rolls/buns        →                yogurt
 {'support': 0.015556685307574987, 'confidence': 0.2746858168761221}

         yogurt,rolls/buns          →              whole milk
 {'support': 0.015556685307574987, 'confidence': 0.4526627218934911}
```

1.3 Dataset II: Custom Transaction List

[9]:
```python
transactions = [
 ["E", "K", "M", "N", "O", "Y"],
 ["D", "E", "K", "N", "O", "Y"],
 ["A", "E", "K", "M"],
 ["L", "K", "M", "U", "Y"],
 ["L", "E", "I", "K", "O"]
]

pprint(transactions)
```

```
[['E', 'K', 'M', 'N', 'O', 'Y'],
 ['D', 'E', 'K', 'N', 'O', 'Y'],
 ['A', 'E', 'K', 'M'],
 ['L', 'K', 'M', 'U', 'Y'],
 ['L', 'E', 'I', 'K', 'O']]
```

1.3.1 Running the Apriori Algorithm

[10]:
```python
support_count = int(input("Enter the minimum support count threshold: "))
print("Minimum Support Count: ", support_count)
frequent_itemsets = apriori(support_count)
```

```
Enter the minimum support count threshold: 2
Minimum Support Count:  2
----------------------------------------------------------------------
Round 2:
        Frequent Itemsets: [['E', 'K'], ['E', 'O'], ['K', 'M'], ['K', 'O'],['K',␣
 ↪'Y']]

Round 3:
        Frequent Itemsets: [['E', 'K', 'O']]
----------------------------------------------------------------------
```

1.3.2 Frequent Sets with maximum frequency

[11]:
```python
frequent_item_analysis(frequent_itemsets,support_count)
```

```
1-Itemsets(5) → (K):5
2-Itemsets(5) → (E,K):4
3-Itemsets(1) → (E,K,O):3
Total number of transactions:  5
```

1.3.3 Finding Associations for a given support and confidence

[14]:
```
min_support = float(input("Enter min support threshold: "))
min_confidence = float(input("Enter min confidence threshold: "))
print_all_rules(frequent_itemsets,min_support = min_support,min_confidence =␣
↪min_confidence)
```

```
Enter min support threshold: 0.6
Enter min confidence threshold: 0.8
                 E                   →                   K
                 {'support': 0.8, 'confidence': 1.0}

                 K                   →                   E
                 {'support': 0.8, 'confidence': 0.8}

                 O                   →                   E
                 {'support': 0.6, 'confidence': 1.0}

                 M                   →                   K
                 {'support': 0.6, 'confidence': 1.0}

                 O                   →                   K
                 {'support': 0.6, 'confidence': 1.0}

                 Y                   →                   K
                 {'support': 0.6, 'confidence': 1.0}

                 O                   →                  E,K
                 {'support': 0.6, 'confidence': 1.0}

                E,O                  →                   K
                 {'support': 0.6, 'confidence': 1.0}

                K,O                  →                   E
                 {'support': 0.6, 'confidence': 1.0}
```

1.3.4 Mining Rules for a given itemset input

[13]:
```
frequent_set = list(map(lambda x: str(x).strip(),input("Enter an itemset␣
 ↪separated by commas: ").split(",")))
print("\033[1mMining rules for:",frequent_set,"\033[0m")
association_rules(frequent_set)
```

```
Enter an itemset separated by commas: E,K,O
Mining rules for: ['E', 'K', 'O']
                 E                   →                 K,O
                 {'support': 0.6, 'confidence': 0.75}

                 K                   →                 E,O
                 {'support': 0.6, 'confidence': 0.6}

                 O                   →                 E,K
                 {'support': 0.6, 'confidence': 1.0}

                E,K                  →                  O
                 {'support': 0.6, 'confidence': 0.75}

                E,O                  →                  K
                 {'support': 0.6, 'confidence': 1.0}

                K,O                  →                  E
                 {'support': 0.6, 'confidence': 1.0}
```